Red Hat Enterprise Linux, Version 7.1

Version:	**0.8**
Status:	**Released**
Last Update:	**2016-09-15**
Classification:	**Red Hat and atsec public**

Trademarks

Legal Notice

Revision History

Revision	Date	Author(s)	Changes to Previous Revision
0.0	2013-08-13	Stephan Mueller	First draft based on RHEL 6.2 ST
0.1	2013-08-30	Stephan Mueller	Corrections in cryptographic specifications
0.2	2013-09-11	Stephan Mueller	Re-adding of Twofish and Serpent
0.3	2014-05-18	Stephan Mueller	Addition of IPsec and NSS DRBG, clarification of mount namespace, replace FIPS 186-3 with 186-4, addition of amtu
0.4	2014-08-20	Stephan Mueller	Remove amtu, clearly define Linux Containers in chapter 7 and update the remainder of the ST accordingly, TLS added, remove all claims that are in addition to OSPP 3.9
0.5	2014-08-25	Stephan Mueller	Sync with ST 0.8, add TSS details to satisfy GPOSPP part 2
0.6	2014-07-22	Stephan Mueller	Address evaluator comments: remove FCS_* SFRs, update FTP_ITC.1, editorial changes.
0.7	2014-09-02	Stephan Mueller	Address evaluator comments
0.8	2014-09-15	Stephan Mueller	Add more details to TSS

Table of Contents

List of Tables

List of Figures

1 Introduction

1.1 Security Target Identification

Title:	Red Hat Enterprise Linux, Version 7.1
Version:	0.8
Status:	Released
Date:	2016-09-15
Sponsor:	Red Hat, Inc.
Developer:	Red Hat, Inc.
Certification Body:	BSI
Certification ID:	BSI-DSZ-CC-0949
Keywords:	Security Target, Common Criteria, Linux Distribution, Embedded Linux

1.2 TOE Identification

The TOE is Red Hat Enterprise Linux Version 7.1.

1.3 TOE Type

The TOE type is a Linux-based general-purpose operating system.

1.4 TOE Overview

1.4.1 Configurations defined with this ST

This security target documents the security characteristics of the Red Hat Enterprise Linux distribution (abbreviated with RHEL throughout this document).

1.4.2 Overview description

Red Hat Enterprise Linux is a highly-configurable Linux-based operating system which has been developed to provide a good level of security as required in commercial environments. It also meets all requirements of the Operating System protection profile [OSPP].

1.4.3 Allowed Unclaimed Functionality

The TOE implements mechanisms without any security claims specified in this Security Target. This section outlines such mechanism which are allowed to be used in the evaluated configuration. As these listed mechanisms may interfere with the operation of the claimed security functionality, the evaluation ensures that the interference does not weaken any security functionality.

RHEL provides virtualization environment based on the Linux KVM technology on x86 architectures. RHEL implements the host system for the virtual machine environment and manages the virtual machines. In addition, RHEL provides management interfaces to administer the virtual machine environment as well as full auditing of user and administrator operations. The virtualization mechanism is not available on the other hardware architectures.

The KVM technology separates the runtime environment of virtual machines from each other. The Linux kernel operates as the hypervisor to the virtual machines but provides a normal computing environment to administrators of the virtual machines. Therefore, the Linux kernel supports the concurrent execution of virtual machines and regular applications. RHEL uses the processor virtualization support to ensure that the virtual machines execute close to the native speed of the hardware.

In addition to the separation of the runtime environment, RHEL also provides system-inherent separation mechanisms to the resources of virtual machines. This separation ensures that the large software component used for virtualizing and simulating devices executing for each virtual machine cannot interfere with each other. Using the SELinux multi-category mechanism, the virtualization and simulation software instances are isolated. The virtual machine management framework configures SELinux multi-category settings transparently to the administrator.

RHEL also provides a strong user space separation mechanism called Linux Containers. These Linux Containers use different kernel mechanism to enforce strong isolation of user space components. The isolation mechanisms include Linux namespaces, Linux control groups and system call filtering.

Using SELinux, RHEL provides a multi-level security framework. The TOE uses mandatory access control together with discretionary and role-based access control. In MLS mode rules are defined to assign sensitivity labels to subjects and objects and to implement the information flow mandatory access control policy modeled based on the concept of Bell-LaPadula.

1.4.4 Compliance with STIG and other standards

The evaluated configuration draws from many standards, including the US STIG standard. It is possible to achieve full compliance with STIG in the evaluated configuration. However, to prevent violation of other configuration standards, the evaluated configuration does not claim full compliance with STIG.

1.4.5 Required Hardware and Software

The following hardware / firmware allows the installation of the TOE:

The following hardware is allowed:

- HP based on x86 64bit Intel Xeon processors:
    - HP Proliant ML series G7, Gen8, Gen9 product line
    - HP Proliant DL series G7, Gen8, Gen9 product line
    - HP ProLiant BL series G7, Gen8, Gen9 product line
    - HP ProLiant SL series G7, Gen8, Gen9 product line
- HP based on AMD64 processors:
    - HP Proliant ML series G7, Gen8 product line
    - HP Proliant DL series G7, Gen8 product line
    - HP ProLiant BL series G7, Gen8 product line
    - HP ProLiant SL series G7, Gen8 product line
- Dell based on x86 64bit Intel:
    - Dell PowerEdge R920
    - Dell PowerEdge R930
    - Dell PowerEdge M620, M520, M420,

  - Dell PowerEdge T430, T630, R430, R530, R630, R730, R730xd, M630, M830, FC430, FC630, FC830, C6320, and Precision R7910

- IBM System p based on Power 8 processors providing execution environments with PowerVM:
  - Big Endian with PowerVM: Tuleta BE model number - Power 835 model 8286-41A
  - Little Endian with RHEV for Power 3.6: Power 835 model 8284-22A
- IBM System z based on z/Architecture processors:
  - zEnterprise EC12 (zEC12)
  - zEnterprise BC12 (zBC12)
  - zEnterprise 196 (z196)
  - zEnterprise 114 (z114)

The following virtual environment is allowed as an execution environment for the TOE:

- KVM on x86 hardware as provided by RHEL 7 or later
- KVM on POWER LE hardware as provided by RHEV-H 3.6 or later

All hardware must be configured using a RAM with automated error correction mechanism present. For example ECC RAM would be suitable to cover that requirement.

1.4.6 Intended Method of Use

1.4.6.1 General-purpose computing environment

The TOE is a Linux-based multi-user multi-tasking operating system. The TOE may provide services to several users at the same time. After successful login, the users have access to a general computing environment, allowing the start-up of user applications, issuing user commands at shell level, creating and accessing files. The TOE provides adequate mechanisms to separate the users and protect their data. Privileged commands are restricted to administrative users.

The TOE is intended to operate in a networked environment with other instantiations of the TOE as well as other well-behaved peer systems operating within the same management domain. All those systems need to be configured in accordance with a defined common security policy.

It is assumed that responsibility for the safeguarding of the user data protected by the TOE can be delegated to human users of the TOE if such users are allowed to log on and spawn processes on their behalf. All user data is under the control of the TOE. The user data is stored in named objects, and the TOE can associate a description of the access rights to that object with each named object.

The TOE enforces controls such that access to data objects can only take place in accordance with the access restrictions placed on that object by its owner, and by administrative users. Ownership of named objects may be transferred under the control of the access control policies implemented by RHEL.

Discretionary access rights (e.g. read, write, execute) can be assigned to data objects with respect to subjects identified with their UID, GID and supplemental GIDs. Once a subject is granted access to an object, the content of that object may be used freely to influence other objects accessible to this subject.

1.4.6.2 Operating Environment

The TOE permits one or more processors and attached peripheral and storage devices to be used by multiple applications assigned to different UIDs to perform a variety of functions requiring controlled shared access to the data stored on the system. With different UIDs proper access restrictions to resources assigned to processes can be enforced using the access control mechanisms provided by the TOE. Such installations and usage scenarios are typical for systems accessed by processes or users local to, or with otherwise protected access to, the computer system.

Note: The TOE provides the platform for installing and running arbitrary services. These additional services are not part of the TOE. The TOE is solely the operating system which provides the runtime environment for such services.

All human users, if existent, as well as all services offered by RHEL are assigned unique user identifiers within the single host system that forms the TOE. This user identifier is used together with the attributes and roles assigned to the user identifier as the basis for access control decisions. Except for virtual machine accesses, the TOE authenticates the claimed identity of the user before allowing the user to perform any further actions. Services may be spawned by the TOE without the need for user-interaction. The TOE internally maintains a set of identifiers associated with processes, which are derived from the unique user identifier upon login of the user or from the configured user identifier for a TOE-spawned service. Some of those identifiers may change during the execution of the process according to a policy implemented by the TOE.

1.4.7 Major Security Features

The primary security features of the TOE are specified as part of the section 1.5.2.2 logical boundary description.

These primary security features are supported by domain separation and reference mediation, which ensure that the features are always invoked and cannot be bypassed.

1.5 TOE Description

1.5.1 Introduction

Red Hat Enterprise Linux is a general purpose, multi-user, multi-tasking Linux based operating system. It provides a platform for a variety of applications.

The RHEL evaluation covers a potentially distributed network of systems running the evaluated versions and configurations of RHEL as well as other peer systems operating within the same management domain. The hardware platforms selected for the evaluation consist of machines which are available when the evaluation has completed and to remain available for a substantial period of time afterwards.

The TOE Security Functions (TSF) consist of functions of RHEL that run in kernel mode plus some trusted processes. These are the functions that enforce the security policy as defined in this Security Target. Tools and commands executed in user mode that are used by an administrative user need also to be trusted to manage the system in a secure way. But as with other operating system evaluations they are not considered to be part of this TSF.

The hardware, the BootProm or BIOS firmware and potentially other firmware layers between the hardware and the TOE are considered to be part of the TOE environment.

The TOE includes standard networking applications, including applications allowing access of the TOE via cryptographically protected communication channels, such as SSH.

System administration tools include the standard command line tools. A graphical user interface for system administration or any other operation is not included in the evaluated configuration.

The TOE environment also includes applications that are not evaluated, but are used as unprivileged tools to access public system services. For example a network server using a port above 1024 may be used as a normal application running without root privileges on top of the TOE. The additional documentation specific for the evaluated configuration provides guidance how to set up such applications on the TOE in a secure way.

1.5.2 TOE boundaries

1.5.2.1 Physical

The Target of Evaluation is based on the following system software:

- Red Hat Enterprise Linux in the above mentioned version

The TOE and its documentation are supplied on ISO images distributed via the Red Hat Network. The TOE includes a package holding the additional user and administrator documentation.

In addition to the installation media, the following documentation is provided:

- Evaluated Configuration Guide published by Red Hat at the end of the evaluation
- Manual pages for all applications, configuration files and system calls

The hardware applicable to the evaluated configuration is listed above. The analysis of the hardware capabilities as well as the firmware functionality is covered by this evaluation to the extent that the following capabilities supporting the security functionality are analyzed and tested:

- Memory separation capability
- Unavailability of privileged processor states to untrusted user code (like the hypervisor state or the SMM)
- Full testing of the security functionality on all listed hardware systems

1.5.2.2 Logical

The primary security features of the TOE are enumerated as follows:

Auditing

The Lightweight Audit Framework (LAF) is designed to be an audit system making Linux compliant with the requirements from Common Criteria. LAF is able to intercept all system calls as well as retrieving audit log entries from privileged user space applications. The subsystem allows configuring the events to be actually audited from the set of all events that are possible to be audited.

The TOE can be deployed as an audit server that receives audit logs from other TOE instances. These audit logs are stored locally. The TOE provides search and review facilities to authorized administrators for all audit logs.

Trusted Channel

The TOE provides cryptographically secured communication to allow remote entities to log into the TOE. For interactive usage, the SSHv2 protocol is provided. The TOE provides the server side as well as the client side applications. Using OpenSSH, password-based and public-key-based authentication are allowed.

Network Information Flow Control

The TOE provides a stateless and stateful packet filter for regular IP-based communication. OSI Layer 3 (IP) and OSI layer 4 (TCP, UDP, ICMP) network protocols can be controlled using this packet filter. To allow virtual machines to communicate with the environment, the TOE provides a bridging functionality. Ethernet frames routed through bridges are controlled by a separate packet filter which implements a stateless packet filter for the TCP/IP protocol family.

The packet filtering functionality offered by the TOE is hooked into the TCP/IP stack of the kernel at different locations. Based on these locations, different filtering capabilities are applicable. The lower level protocols are covered by the EBTables filter mechanism which includes the filtering of Ethernet frames including the ARP layer -- EBTables is not covered in this evaluation. The higher level protocols of TCP/IP are covered with the IPTables mechanism which allows filtering of IP and TCP, UDP, ICMP packets. In addition, IPTables offers a stateful packet filter for the mentioned higher level protocols.

Identification and Authentication

User identification and authentication in the TOE includes all forms of interactive login (e.g. using the SSH protocol or log in at the local console) as well as identity changes through the su or sudo command. These all rely on explicit authentication information provided interactively by a user.

The authentication security function allows password-based authentication. For SSH access, public-key-based authentication is also supported.

Password quality enforcement mechanisms are offered by the TOE which are enforced at the time when the password is changed.

The TOE provides a framework to authenticate with remote servers, such as LDAP, Kerberos or Microsoft Windows Active Directory. The SSSD daemon establishes the connection to the remote authentication stores and provides a local authentication cache in case the connection is severed. SSSD is integrated with the Linux authentication mechanism by using a PAM module.

Discretionary Access Control

DAC allows owners of named objects to control the access permissions to these objects. These owners can permit or deny access for other users based on the configured permission settings. The DAC mechanism is also used to ensure that untrusted users cannot tamper with the TOE mechanisms.

In addition to the standard Unix-type permission bits for file system objects as well as IPC objects, the TOE implements POSIX access control lists. These ACLs allow the specification of the access to individual file system objects down to the granularity of a single user.

Security Management

The security management facilities provided by the TOE are usable by authorized users and/or authorized administrators to modify the configuration of TSF.

The TOE allows remote management via OpenSSH. Administrative users can log in remotely and perform the same management tasks as a locally operating administrator.

1.5.2.3 Configurations

The evaluated configurations are defined as follows:

- The CC evaluated package set must be selected at install time in accordance with the description provided in the Evaluated Configuration Guide and installed accordingly.
- The TOE supports the use of IPv4 and IPv6, both are also supported in the evaluated configuration. IPv6 conforms to the following RFCs:
    - RFC 2460 specifying the basic IPv6 protocol
    - IPv6 source address selection as documented in RFC 3484
    - Linux implements several new socket options (IPV6_RECVPKTINFO, IPV6_PKTINFO, IPV6_RECVHOPOPTS, IPV6_HOPOPTS, IPV6_RECVDSTOPTS, IPV6_DSTOPTS, IPV6_RTHDRDSTOPTS, IPV6_RECVRTHDR, IPV6_RTHDR, IPV6_RECVHOPOPTS, IPV6_HOPOPTS, IPV6_{RECV,}TCLASS) and ancillary data in order to support advanced IPv6 applications including ping, traceroute, routing daemons and others. The following section introduces Internet Protocol Version 6 (IPv6). For additional information about referenced socket options and advanced IPv6 applications, see RFC 3542
    - Transition from IPv4 to IPv6: dual stack, and configured tunneling according to RFC 4213.
    - Additional RFCs covering various cryptographic aspects are outlined as part of the Security Functional Requirements.
- The default configuration for identification and authentication are the defined password-based PAM modules as well as by the certificate based authentication for OpenSSH. Support for other authentication options, e.g. smart card authentication, is not included in the evaluation configuration.
- If the system console is used, it must be connected directly to the TOE and afforded the same physical protection as the TOE.

Deviations from the configurations and settings specified with the Evaluated Configuration Guide are not permitted.

The TOE comprises a single system (and optional peripherals) running the TOE software listed. Cluster configurations touching the state information of security functions are not permitted in the evaluated configuration. This means it is permissible to install applications which by themselves offer cluster functionality covering their state, such as JBoss EAP.

1.5.2.4 TOE Environment

Several TOE systems may be interlinked in a network, and individual networks may be joined by bridges and/or routers, or by TOE systems which act as routers and/or gateways. Each of the TOE systems implements its own security policy. The TOE does not include any synchronization function for those policies. As a result a single user may have user accounts on each of those systems with different UIDs, different roles, and other different attributes. (A synchronization method may optionally be used, but it not part of the TOE and must not use methods that conflict with the TOE requirements.)

If other systems are connected to a network they need to be configured and managed by the same authority using an appropriate security policy that does not conflict with the security policy of the TOE. All connections between this network and untrusted networks (e. g. the Internet) need to be protected by appropriate measures such as carefully configured firewall systems that prohibit attacks from the untrusted networks. Those protections are part of the TOE environment.

1.5.2.5 Security Policy Model

The security policy for the TOE is defined by the security functional requirements in chapter 6. The following is a list of the subjects and objects participating in the policy.

Subjects:

- Processes acting on behalf of a human user or technical entity.
- Processes acting on behalf of a human user or technical entity providing a virtual machine environment.

Named objects:

- File system objects in the following allowed file systems:
  - XFS - standard file system for general data
  - VFAT - special purpose file system for UEFI BIOS support mounted at /boot/efi
  - Ext4 - standard file system for general data
  - iso9660 - ISO9660 file system for CD-ROM and DVD
  - tmpfs - the temporary file system backed by RAM
  - rootfs - the virtual root file system used temporarily during system boot
  - procfs - process file system holding information about processes, general statistical data and tunable kernel parameters
  - sysfs - system-related file system covering general information about resources maintained by the kernel including several tunable parameters for these resources
  - devpts - pseudoterminal file system for allocating virtual TTYs on demand
  - devtmpfs - temporary file system that allows the kernel to generate character or block device nodes
  - binfmt_misc - configuration interface allowing the assignment of executable file formats with user space applications
  - securityfs - interface for loadable security modules (LSM) to provide tunables and configuration interfaces to user space
  - selinuxfs - interface for allowing user space components to interact with the SELinux module inside the kernel, including managing the SELinux policy.

  Note that the TOE supports a number of additional virtual (i.e. without backing of persistent storage) file systems which are only accessible to the TSF - they are not or cannot be mounted. All above mentioned virtual file systems implement access decisions based DAC attributes inferred from the underlying process’ DAC attributes. Additional restrictions may apply for specific objects in this file system.
- Inter Process Communication (IPC) objects:
  - Semaphores
  - Shared memory
  - Message queues
  - Named pipes

  - UNIX domain socket special files

TSF data:

- TSF executable code
- Subject meta data - all data used for subjects except data which is not interpreted by the TSF and does not implement parts of the TSF (this data is called user data)
- Named object meta data - all data used for the respective objects except data which is not interpreted by the TSF and does not implement parts of the TSF (this data is called user data)
- User accounts, including the security attributes defined by FIA_ATD.1
- Audit records

User data:

- Non-TSF executable code used to drive the behavior of subjects
- Data not interpreted by TSF and stored or transmitted using named objects

1.5.3 Additional Functions

The TOE provides many more functions and mechanisms. The evaluation ensures that all these additional functions do not interfere with the above mentioned security mechanisms in the evaluated configuration. The mechanisms given in the following list, however, may interfere with the security functionality of the TOE and should be allowed in the evaluated configuration. Therefore, the evaluation assesses the functionality to verify that the impact on the security functionality at most adds further restrictions as outlined below.

- KVM virtualization support: The TOE offers virtualization support via KVM. That virtualization support shall be allowed to be used such that it does not interfere with the operation of the security functions. The evaluation ensures that the constraints associated with the use of KVM in the evaluated configuration guide has no adverse impact on the security functionality. In addition, the libvirt daemon is allowed to run with the privileges of the root user to allow management of KVM.
- Linux Container: Linux Container provide execution environments for processes. These Linux Containers isolate the processes, ensure resource accounting and limitation as well as Linux kernel service limitation. The evaluation ensures that only additional restrictions are enforced with Linux Containers. Therefore, Linux Containers cannot be used to override security mechanisms claimed in this ST.
- SELinux: SELinux together with a SELinux policy may enforce additional access restrictions of applications to resources and objects. The evaluation ensures that only additional restrictions are enforced with SELinux. Therefore, SELinux cannot be used to override security mechanisms claimed in this ST. Note, the MLS mechanism is implemented using a specifically crafted SELinux policy.
- IPSec VPN: The TOE offers IPSec VPN where the IPSEC protocol is implemented in the kernel and the IKE protocol in a user space daemon. When applying an IPSec VPN, the communicated data is automatically encrypted or decrypted by the operating system.

Additional mechanisms and functions that would interfere with the operation of the security functions are disallowed in the evaluated configuration and the Evaluation Configuration Guide provides instructions to the administrator on how to disable them. Note: TOE mechanism which provide additional restrictions to the above claimed security functions are allowed in the evaluated configuration. For example, the eCryptFS cryptographic file system provided with the TOE and permitted in the evaluated configuration even though they have not been subject to this evaluation.

The eCryptFS provides further restrictions on, for example, the security function of discretionary access control mechanism for file system objects and therefore cannot breach the security functionality as the discretionary access control rules of the "lower" file system are still enforced. The following table enumerates mechanisms that are provided with the TOE but which are excluded from the evaluation:

Functions	**Exclusion discussion**
eCryptFS	eCryptFS is allowed to be used in the evaluated configuration. The encryption capabilities provided with this file system is, however, not subject to this evaluation.
dm-crypt	dm-crypt is allowed to be used in the evaluated configuration. The encryption capabilities provided with this block device encryption mechanism is, however, not subject to this evaluation.
SMACK	The mandatory access control functionality offered by the SMACK LSM is not assessed by the evaluation and disabled in the evaluated configuration. The SELinux LSM provides the mandatory access control policy enforcement.
SSL / TLS tunnels	The TOE provides the stunnel application which can be used to establish SSL and TLS tunnels with remote peers. This application however was excluded from evaluation assessment.
GSS-API Security Mechanisms	The GSS-API is used to secure the connection between different audit daemons. The security mechanisms used by the GSS-API, however, is not part of the evaluation. Therefore, A.CONNECT applies to the audit-related communication link.
Runtime protection mechanisms	The Linux kernel provides several runtime protection mechanisms to lower the probability of a successful exploitation of typical software errors like buffer overruns. This mechanism, however, is not part of the evaluation.

Table 1: Non-evaluated functionalities

Note: Packages and mechanisms not covered with security claims and subsequent assessments during the evaluation or disabling the respective functionality in the evaluated configuration result from resource constraints during the evaluation but does not imply that the respective package or functionality is implemented insecurely.

2 CC Conformance Claim

This Security Target is CC Part 2 extended and CC Part 3 conformant.

This Security Target claims conformance to the following Protection Profiles and PP packages:

- [OSPP]: General-Purpose Operating System Protection Profile. Version 3.9 as of 2012-12-06; strict conformance.

Common Criteria [CC] version 3.1 revision 4 is the basis for this conformance claim.

3 Security Problem Definition

3.1 Threat Environment

Threats to be countered by the TOE are characterized by the combination of an asset being subject to a threat, a threat agent and an adverse action.

The definition of threat agents and protected assets that follows is derived from the OSPP.

3.1.1 Assets

Assets to be protected are:

- Storage objects used to store user data and/or TSF data, where this data needs to be protected from any of the following operations:
    - Unauthorized read access
    - Unauthorized modification
    - Unauthorized deletion of the object
    - Unauthorized creation of new objects
    - Unauthorized management of object attributes
- TSF functions and associated TSF data
- The resources managed by the TSF that are used to store the above-mentioned objects, including the metadata needed to manage these objects.

3.1.2 Threat Agents

Threat agents are external entities that potentially may attack the TOE. They satisfy one or more of the following criteria:

- External entities not authorized to access assets may attempt to access them either by masquerading as an authorized entity or by attempting to use TSF services without proper authorization.
- External entities authorized to access certain assets may attempt to access other assets they are not authorized to either by misusing services they are allowed to use or by masquerading as a different external entity.
- Untrusted subjects may attempt to access assets they are not authorized to either by misusing services they are allowed to use or by masquerading as a different subject.

Threat agents are typically characterized by a number of factors, such as expertise, available resources, and motivation, with motivation being linked directly to the value of the assets at stake. The TOE protects against intentional and unintentional breach of TOE security by attackers possessing an enhanced-basic attack potential.

3.1.3 Threats countered by the TOE

T.ACCESS.TSFDATA

A threat agent might read or modify TSF data using functions of the TOE without the necessary authorization.

T.ACCESS.USERDATA

A threat agent might gain access to user data stored, processed or transmitted by the TOE without being appropriately authorized according to the TOE security policy by using functions provided by the TOE.

T.ACCESS.TSFFUNC

A threat agent might use or manage functionality of the TSF bypassing protection mechanisms of the TSF.

T.ACCESS.COMM

A threat agent may access cryptographically protected data transferred via a trusted channel between the TOE and another remote trusted IT system, modify such data during transfer in a way not detectable by the receiving party or masquerade as a remote trusted IT system.

T.RESTRICT.NETTRAFFIC

A threat agent may send data packets to the recipient in the TOE via a network communication channel in violation of the information flow control policy.

T.IA.MASQUERADE

A threat agent may masquerade as an authorized entity including the TOE itself or a part of the TOE in order to gain unauthorized access to user data, TSF data, or TOE resources.

T.IA.USER

A threat agent may gain access to user data, TSF data or TOE resources with the exception of public objects without being identified and authenticated by the TSF.

T.UNATTENDED_SESSION

A threat agent may gain unauthorized access to an unattended session.

3.2 Assumptions

3.2.1 Environment of use of the TOE

3.2.1.1 Physical

A.PHYSICAL

It is assumed that the IT environment provides the TOE with appropriate physical security, commensurate with the value of the IT assets protected by the TOE.

3.2.1.2 Personnel

A.MANAGE

The TOE security functionality is managed by one or more competent individuals. The system administrative personnel are not careless, willfully negligent, or hostile, and will follow and abide by the instructions provided by the guidance documentation.

A.AUTHUSER

Authorized users possess the necessary authorization to access at least some of the information managed by the TOE and are expected to act in a cooperating manner in a benign environment.

A.TRAINEDUSER

Users are sufficiently trained and trusted to accomplish some task or group of tasks within a secure IT environment by exercising complete control over their user data.

3.2.1.3 Procedural

A.DETECT

Any modification or corruption of security-enforcing or security-relevant files of the TOE, user or the underlying platform caused either intentionally or accidentally will be detected by an administrative user.

A.PEER.MGT

All remote trusted IT systems trusted by the TSF to provide TSF data or services to the TOE, or to support the TSF in the enforcement of security policy decisions are assumed to be under the same management control and operate under security policy constraints compatible with those of the TOE.

A.PEER.FUNC

All remote trusted IT systems trusted by the TSF to provide TSF data or services to the TOE, or to support the TSF in the enforcement of security policy decisions are assumed to correctly implement the functionality used by the TSF consistent with the assumptions defined for this functionality.

3.2.1.4 Connectivity

A.CONNECT

All connections to and from remote trusted IT systems and between physically-separate parts of the TSF not protected by the TSF itself are physically or logically protected within the TOE environment to ensure the integrity and confidentiality of the data transmitted and to ensure the authenticity of the communication end points.

3.3 Organizational Security Policies

P.ACCOUNTABILITY

The users of the TOE shall be held accountable for their security-relevant actions within the TOE.

P.USER

Authority shall only be given to users who are trusted to perform the actions correctly.

P.ROLES

Administrative authority to TSF functionality shall be given to trusted personnel and be as restricted as possible supporting only the administrative duties the person has.

4 Security Objectives

4.1 Objectives for the TOE

O.AUDITING

The TSF must be able to record defined security-relevant events (which usually include security-critical actions of users of the TOE). The TSF must protect this information and present it to authorized users if the audit trail is stored on the local system. The information recorded for security-relevant events must contain the time and date the event happened and, if possible, the identification of the user that caused the event, and must be in sufficient detail to help the authorized user detect attempted security violations or potential misconfiguration of the TOE security features that would leave the IT assets open to compromise.

O.DISCRETIONARY.ACCESS

The TSF must control access of subjects and/or users to named resources based on identity of the object. The TSF must allow authorized users to specify for each access mode which users/subjects are allowed to access a specific named object in that access mode.

O.NETWORK.FLOW

The TOE shall mediate network communication between an entity outside of the TOE and a recipient within the TOE in accordance with its network information flow security policy.

O.SUBJECT.COM

The TOE shall mediate any possible sharing of objects or resources between subjects acting with different subject security attributes in accordance with its discretionary access control policy.

O.I&A

The TOE must ensure that users have been successfully authenticated before allowing any action the TOE has defined to provide to authenticated users only.

O.MANAGE

The TSF must provide all the functions and facilities necessary to support the authorized users that are responsible for the management of TOE security mechanisms, must allow restringing such management actions to dedicated users, and must ensure that only such authorized users are able to access management functionality.

O.TRUSTED_CHANNEL

The TSF must allow authorized users to remotely access the TOE using a cryptographically-protected network protocol that ensures integrity and confidentiality of the transported data and is able to authenticate the end points of the communication. Note that the same protocols may also be used in the case where the TSF is physically separated into multiple parts that must communicate securely with each other over untrusted network connections. The protocol must also prevent masquerading of the remote trusted IT system.

O.UNATTENDED_SESSION

The TOE must allow for the temporary suspension of a user's session allowing the continuation of such a suspended session and user related input and output only after the user has resumed the session by re-authenticating himself to the TSF.

4.2 Objectives for the Operational Environment

OE.ADMIN

Those responsible for the TOE are competent and trustworthy individuals, capable of managing the TOE and the security of the information it contains.

OE.REMOTE

If the TOE relies on remote trusted IT systems to support the enforcement of its policy, those systems provide the functions required by the TOE and are sufficiently protected from any attack that may cause those functions to provide false results.

OE.INFO_PROTECT

Those responsible for the TOE must establish and implement procedures to ensure that information is protected in an appropriate manner. In particular:

- All network and peripheral cabling must be approved for the transmittal of the most sensitive data held by the system. Such physical links are assumed to be adequately protected against threats to the confidentiality and integrity of the data transmitted.
- DAC protections on security-relevant files (such as audit trails and authentication databases) shall always be set up correctly.
- Users are authorized to access parts of the data managed by the TOE and are trained to exercise control over their own data.

OE.INSTALL

Those responsible for the TOE must establish and implement procedures to ensure that the hardware, software and firmware components that comprise the system are distributed, installed and configured in a secure manner supporting the security mechanisms provided by the TOE.

OE.MAINTENANCE

Authorized users of the TOE must ensure that the comprehensive diagnostics facilities provided by the product are invoked at every scheduled preventative maintenance period.

OE.PHYSICAL

Those responsible for the TOE must ensure that those parts of the TOE critical to enforcement of the security policy are protected from physical attack that might compromise IT security objectives. The protection must be commensurate with the value of the IT assets protected by the TOE.

OE.RECOVER

Those responsible for the TOE must ensure that procedures and/or mechanisms are provided to assure that after system failure or other discontinuity, recovery without a protection (security) compromise is achieved.

OE.TRUSTED.IT.SYSTEM

The remote trusted IT systems implement the protocols and mechanisms required by the TSF to support the enforcement of the security policy.

These remote trusted IT systems are under the same management domain as the TOE, are managed based on the same rules and policies applicable to the TOE, and are physically and logically protected equivalent to the TOE.

4.3 Security Objectives Rationale

4.3.1 Coverage

The following table provides a mapping of TOE objectives to threats and policies, showing that each objective counters or enforces at least one threat or policy, respectively.

Objective	Threats / OSPs
O.AUDITING	P.ACCOUNTABILITY
O.DISCRETIONARY.ACCESS	T.ACCESS.TSFDATA T.ACCESS.USERDATA
O.NETWORK.FLOW	T.RESTRICT.NETTRAFFIC
O.SUBJECT.COM	T.ACCESS.TSFDATA T.ACCESS.USERDATA
O.I&A	T.IA.MASQUERADE T.IA.USER
O.MANAGE	T.ACCESS.TSFFUNC P.ACCOUNTABILITY P.USER P.ROLES
O.TRUSTED_CHANNEL	T.ACCESS.TSFDATA T.ACCESS.USERDATA T.ACCESS.TSFFUNC T.ACCESS.COMM
O.UNATTENDED_SESSION	T.UNATTENDED_SESSION

Table 2: Mapping of security objectives to threats and policies

The following table provides a mapping of the objectives for the Operational Environment to assumptions, threats and policies, showing that each objective holds, counters or enforces at least one assumption, threat or policy, respectively.

Objective	Assumptions / Threats / OSPs
OE.ADMIN	A.MANAGE A.AUTHUSER A.TRAINEDUSER P.ROLES
OE.REMOTE	A.CONNECT T.ACCESS.COMM
OE.INFO_PROTECT	A.PHYSICAL A.MANAGE A.AUTHUSER A.TRAINEDUSER P.USER
OE.INSTALL	A.MANAGE A.DETECT
OE.MAINTENANCE	A.DETECT
OE.PHYSICAL	A.PHYSICAL
OE.RECOVER	A.MANAGE A.DETECT
OE.TRUSTED.IT.SYSTEM	A.PEER.MGT A.PEER.FUNC A.CONNECT

Table 3: Mapping of security objectives for the Operational Environment to assumptions, threats and policies

4.3.2 Sufficiency

The following rationale provides justification that the security objectives are suitable to counter each individual threat and that each security objective tracing back to a threat, when achieved, actually contributes to the removal, diminishing or mitigation of that threat.

Threat	Rationale for security objectives
T.ACCESS.TSFDATA	The threat of accessing TSF data without proper authorization is mitigated by: ● O.TRUSTED_CHANNEL requiring cryptographically-protected communication channels for data including TSF data controlled by the TOE in transit between trusted IT systems, ● O.DISCRETIONARY.ACCESS requiring that data, including TSF data stored with the TOE, have discretionary access control protection, ● O.SUBJECT.COM requiring the TSF to mediate communication between subjects.
T.ACCESS.USERDATA	The threat of accessing user data without proper authorization is mitigated by:

Threat	Rationale for security objectives
	● O.TRUSTED_CHANNEL requiring cryptographically-protected communication channels for data including user data controlled by the TOE in transit between trusted IT systems, ● O.DISCRETIONARY.ACCESS requiring that data including user data stored with the TOE, have discretionary access control protection, ● O.SUBJECT.COM requiring the TSF to mediate communication between subjects.
T.ACCESS.TSFFUNC	The threat of accessing TSF functions without proper authorization is mitigated by: ● O.TRUSTED_CHANNEL requiring cryptographically-protected communication channels to limit which TSF functions are accessible to external entities, ● O.MANAGE requiring that only authorized users utilize management TSF functions.
T.ACCESS.COMM	The threat of accessing a communication channel that establishes a trust relationship between the TOE and another remote trusted IT system is mitigated by: ● O.TRUSTED_CHANNEL requiring that the TOE implements a trusted channel between itself and a remote trusted IT system protecting the user data and TSF data transferred over this channel from disclosure and undetected modification and prevents masquerading of the remote trusted IT system, ● OE.REMOTE requiring that those systems providing the functions required by the TOE are sufficiently protected from any attack that may cause those functions to provide false results.
T.RESTRICT.NETTRAFFIC	The threat of accessing information or transmitting information to other recipients via network communication channels without authorization for this communication attempt is mitigated by: ● O.NETWORK.FLOW requiring the TOE to mediate the communication between itself and remote entities in accordance with its security policy.
T.IA.MASQUERADE	The threat of masquerading as an authorized entity in order to gain unauthorized access to user data, TSF data or TOE resources is mitigated by: ● O.I&A requiring that each entity interacting with the TOE is properly identified and authenticated before allowing any action the TOE is defined to provide to authenticated users only.
T.IA.USER	The threat of accessing user data, TSF data or TOE resources without being identified and authenticated is removed by: ● O.I&A requiring that each entity interacting with the TOE is properly identified and authenticated before allowing any action the TOE has defined to provide to authenticated users only.
T.UNATTENDED_SESSION	The threat of an attack agent using an unattended session to gain access to protected functionality of the TSF, user data, or TSF data is mitigated:

Threat	Rationale for security objectives
	● O.UNATTENDED_SESSION requiring the capability that unattended sessions can be protected from use by unauthorized persons.

Table 4: Sufficiency of objectives countering threats

The following rationale provides justification that the security objectives for the environment are suitable to cover each individual assumption, that each security objective for the environment that traces back to an assumption about the environment of use of the TOE, when achieved, actually contributes to the environment achieving consistency with the assumption, and that if all security objectives for the environment that trace back to an assumption are achieved, the intended usage is supported.

Assumption	Rationale for security objectives
A.PHYSICAL	The assumption on the IT environment to provide the TOE with appropriate physical security, commensurate with the value of the IT assets protected by the TOE is covered by: ● OE.INFO_PROTECT requiring the approval of network and peripheral cabling, ● OE.PHYSICAL requiring physical protection.
A.MANAGE	The assumptions on the TOE security functionality being managed by one or more trustworthy individuals is covered by: ● OE.ADMIN requiring trustworthy personnel managing the TOE, ● OE.INFO_PROTECT requiring personnel to ensure that information is protected in an appropriate manner, ● OE.INSTALL requiring personnel to ensure that components that comprise the system are distributed, installed and configured in a secure manner supporting the security mechanisms provided by the TOE, ● OE.RECOVER requiring personnel to assure that after system failure or other discontinuity, recovery without a protection (security) compromise is achieved.
A.AUTHUSER	The assumption on authorized users to possess the necessary authorization to access at least some of the information managed by the TOE and to act in a cooperating manner in a benign environment is covered by: ● OE.ADMIN ensuring that those responsible for the TOE are competent and trustworthy individuals, capable of managing the TOE and the security of the information it contains. ● OE.INFO_PROTECT requiring that DAC protections on security-relevant files (such as audit trails and authentication databases) shall always be set up correctly and that users are authorized to access parts of the data maintained by the TOE.
A.TRAINEDUSER	The assumptions on users to be sufficiently trained and trusted to accomplish some task or group of tasks within a secure IT environment by exercising complete control over their user data is covered by:

Assumption	Rationale for security objectives
	● OE.ADMIN requiring competent personnel managing the TOE. ● OE.INFO_PROTECT requiring that those responsible for the TOE must establish and implement procedures to ensure that information is protected in an appropriate manner and that users are trained to exercise control over their own data.
A.DETECT	The assumption that modification or corruption of security-enforcing or security-relevant files will be detected by an administrative user is covered by: ● OE.INSTALL requiring an administrative user to ensure that the TOE is distributed, installed and configured in a secure manner supporting the security mechanisms provided by the TOE. ● OE.MAINTENANCE requiring an administrative user to ensure that the diagnostics facilities are invoked at every scheduled preventative maintenance period, verifying the correct operation of the TOE. ● OE.RECOVER requiring an administrative user to ensure that procedures and/or mechanisms are provided to assure that after system failure or other discontinuity, recovery without a protection (security) compromise is achieved.
A.PEER.MGT	The assumption on all remote trusted IT systems to be under the same management control and operate under security policy constraints compatible with those of the TOE is covered by: ● OE.TRUSTED.IT.SYSTEM requiring that these remote trusted IT systems are under the same management domain as the TOE, and are managed based on the same rules and policies applicable to the TOE.
A.PEER.FUNC	The assumption on all remote trusted IT systems to correctly implement the functionality used by the TSF consistent with the assumptions defined for this functionality is covered by: ● OE.TRUSTED.IT.SYSTEM requiring that the remote trusted IT systems implement the protocols and mechanisms required by the TSF to support the enforcement of the security policy.
A.CONNECT	The assumption on all connections to and from remote trusted IT systems and between physically separate parts of the TSF not protected by the TSF itself are physically or logically protected is covered by: ● OE.REMOTE requiring that remote trusted IT systems provide the functions required by the TOE and are sufficiently protected from any attack that may cause those functions to provide false results. ● OE.TRUSTED.IT.SYSTEM demanding the physical and logical protection equivalent to the TOE.

Table 5: Sufficiency of objectives holding assumptions

The following rationale provides justification that the security objectives are suitable to cover each individual organizational security policy (OSP), that each security objective that traces back to an OSP, when achieved, actually contributes to the implementation of the OSP, and that if all security objectives that trace back to an OSP are achieved, the OSP is implemented.

OSP	Rationale for security objectives
P.ACCOUNTABILITY	The policy to hold users accountable for their security-relevant actions within the TOE is implemented by: ● O.AUDITING providing the TOE with audit functionality, ● O.MANAGE allowing the management of this function.
P.USER	The policy to match the trust given to a user and the actions the user is given authority to perform is implemented by: ● O.MANAGE allowing appropriately-authorized users to manage the TSF, ● OE.INFO_PROTECT, which requires that users are trusted to use the protection mechanisms of the TOE to protect their data.
P.ROLES	The policy to only give trusted users authority is implemented by: ● O.MANAGE allowing appropriately-authorized users to manage the TSF. ● OE.ADMIN, which requires that users responsible for the TOE are competent and trustworthy individuals, capable of managing the TOE and the security of the information it contains.

Table 6: Sufficiency of objectives enforcing Organizational Security Policies

5 Extended Components Definition

The Security Target uses the extended components of FIA_PK_EXT.1 as well as FMT_SMF_RMT.1 defined by [OSPP]. They are not re-defined here again.

6 Security Requirements

6.1 TOE Security Functional Requirements

All of the following SFRs are derived from the OSPP supplemented with additional SFRs for add-on functionality.

The following table shows the SFRs for the TOE, and the operations performed on the components according to CC part 1: iteration (Iter.), refinement (Ref.), assignment (Ass.) and selection (Sel.).

Security functional group	Security functional requirement	Base security functional component	Source	Operations			
				Iter.	Ref.	Ass.	Sel.
General-purpose computing environment	FAU_GEN.1 Audit data generation		OSPP	No	Yes	No	No
	FAU_GEN.2 User identity association		OSPP	No	No	No	No
	FAU_SAR.1 Audit review		OSPP	No	No	Yes	No
	FAU_SAR.2 Restricted audit review		OSPP	No	No	No	No
	FAU_SEL.1 Selective audit		OSPP	No	No	Yes	No
	FAU_STG.1 Protected audit trail storage		OSPP	No	No	No	Yes
	FAU_STG.3 Action in case of possible audit data loss		OSPP	No	Yes	Yes	No
	FAU_STG.4 Prevention of audit data loss		OSPP	No	Yes	Yes	Yes
	FDP_ACC.1(PSO) Subset access control	FDP_ACC.1	OSPP	Yes	No	Yes	No
	FDP_ACC.1(TSO) Subset access control	FDP_ACC.1	OSPP	Yes	No	Yes	No
	FDP_ACF.1(PSO) Security attribute based access control	FDP_ACF.1	OSPP	Yes	No	Yes	No
	FDP_ACF.1(TSO) Security attribute based access control	FDP_ACF.1	OSPP	Yes	No	Yes	No
	FDP_IFC.1 Complete information flow control		OSPP	Yes	No	Yes	No
	FDP_IFF.1 Simple security attributes		OSPP	Yes	Yes	Yes	Yes
	FDP_RIP.2 Full residual information protection		OSPP	No	No	No	Yes
	FIA_AFL.1 Authentication failure handling		OSPP	No	No	Yes	No

Security functional group	Security functional requirement	Base security functional component	Source	Operations			
				Iter.	Ref.	Ass.	Sel.
	FIA_ATD.1 User attribute definition		OSPP	No	No	Yes	No
	FIA_UAU.1(RITE) Timing of authentication	FIA_UAU.1	OSPP	Yes	No	Yes	No
	FIA_UAU.1(HU) Timing of authentication	FIA_UAU.1	OSPP	Yes	No	Yes	No
	FIA_UAU.5 Multiple authentication mechanisms		OSPP	No	No	Yes	Yes
	FIA_UAU.7 Protected authentication feedback		OSPP	No	No	No	No
	FIA_UID.1 Timing of identification		OSPP	No	No	Yes	No
	FIA_USB.1 User-subject binding		OSPP	No	No	Yes	No
	FIA_PK_EXT.1 Public key based authentication		OSPP	No	No	Yes	Yes
	FPT_STM.1 Reliable time stamps		OSPP	No	No	No	No
	FTA_SSL.1 TSF-initiated session locking		OSPP	No	No	Yes	No
	FTA_SSL.2 User-initiated locking		OSPP	No	No	Yes	No
	FTP_ITC.1 Inter-TSF trusted channel		OSPP	No	No	Yes	Yes
Management related functionality	FMT_MOF.1 Management of security functions behaviour		OSPP	Yes	No	Yes	No
	FMT_MSA.1(PSO) Management of object security attributes	FMT_MSA.1	OSPP	Yes	No	Yes	Yes
	FMT_MSA.1(TSO) Management of object security attributes	FMT_MSA.1	OSPP	Yes	No	Yes	No
	FMT_MSA.3(DAC/PSO) Static attribute initialisation	FMT_MSA.3	OSPP	Yes	No	Yes	No
	FMT_MSA.3(DAC/TSO) Static attribute initialisation	FMT_MSA.3	OSPP	Yes	No	Yes	No
	FMT_MSA.3(NI) Static attribute initialisation	FMT_MSA.3	OSPP	Yes	No	Yes	Yes
	FMT_MSA.4 Security attribute value inheritance		OSPP	No	No	Yes	No
	FMT_MTD.1(AE) Management of TSF data	FMT_MTD.1	OSPP	Yes	No	Yes	No

Security functional group	Security functional requirement	Base security functional component	Source	Operations			
				Iter.	Ref.	Ass.	Sel.
	FMT_MTD.1(AS) Management of TSF data	FMT_MTD.1	OSPP	Yes	No	Yes	Yes
	FMT_MTD.1(AT) Management of TSF data	FMT_MTD.1	OSPP	Yes	No	Yes	Yes
	FMT_MTD.1(AF) Management of TSF data	FMT_MTD.1	OSPP	Yes	No	Yes	Yes
	FMT_MTD.1(CM) Management of TSF data	FMT_MTD.1	OSPP	Yes	No	Yes	Yes
	FMT_MTD.1(NI) Management of TSF data	FMT_MTD.1	OSPP	Yes	No	Yes	Yes
	FMT_MTD.1(IAT) Management of TSF data	FMT_MTD.1	OSPP	Yes	No	Yes	No
	FMT_MTD.1(IAF) Management of TSF data	FMT_MTD.1	OSPP	Yes	No	Yes	No
	FMT_MTD.1(IAU) Management of TSF data	FMT_MTD.1	OSPP	Yes	Yes	Yes	No
	FMT_REV.1(OBJ) Revocation	FMT_REV.1	OSPP	Yes	No	Yes	No
	FMT_REV.1(USR) Revocation	FMT_REV.1	OSPP	Yes	No	Yes	No
	FMT_SMF_RMT.1 Remote Management Capabilities		OSPP	No	No	No	No
	FMT_SMR.1 Security management roles		OSPP	No	No	Yes	No

Table 7: SFRs for the TOE

6.1.1 General-purpose computing environment

6.1.1.1 Audit data generation (FAU_GEN.1)

FAU_GEN.1.1 The TSF shall be able to generate an audit record of the following auditable events:

a) Start-up and shutdown of the audit functions;

b) All auditable events for the not specified level of audit; and

c) all modifications to the set of events being audited;

d) all user authentication attempts;

e) all denied accesses to objects for which the access control policy defined in the OSPP base applies;

f) explicit modifications of access rights to objects covered by the access control policies; and

g) other specifically defined auditable events as defined in the table in FAU_GEN.1.2.

FAU_GEN.1.2 The TSF shall record within each audit record at least the following information:

a) Date and time of the event, type of event, subject identity (if applicable), and outcome (success or failure) of the event; and

b) for all management SFRs included in the Security Target: the identity of the user that performed/attempted to perform the management operation *consisting of the user identifier and the identifier of the Linux user namespace the user is confined to if applicable,* an identification of what was managed and the indication what the administrative user has changed as part of the management operation, and

c) For each audit event type, based on the auditable event definitions of the functional components included in the following table:

SFR	Events and Event specific information
FAU_SAR.1	Event: Any attempt to access the audit records ● identity of the user attempting to access the audit records ● success or failure
FAU_SEL.1	Event: Any attempt to modify the events to be audited ● identity of the user attempting to modify the events to be audited ● success or failure ● in case of success: modification to the set of events to be audited
FDP_ACF.1(PSO), FDP_ACF.1(TSO)	Event: Any attempt to access an object protected by the SFP ● identity of the user attempting to access an object protected by the SFP. Note: if the operation is attempted by a subject not operating on behalf of a user: identity of the subject ● identity of the object the user attempts to access ● attempted operation ● success or failure
FDP_IFF.1	Event: Denied information flow ● identification of the network interface ● reason for denying information flow

SFR	Events and Event specific information
FIA_AFL.1	Event: Exceeding the limit of unsuccessful consecutive authentication attempts ● user identity where the limit was exceeded
FIA_UAU.1(HU)	Event: Verification that a user has been successfully authenticated ● user identity ● indicator that the user has been successfully authenticated In the case the authentication is performed by the TOE, also the event of a failed authentication attempt needs to be auditable: ● user identity provided ● indicator that the authentication failed
FTA_SSL.1	Event: Re-authentication attempt to unlock a session ● user identity ● success or failure of re-authentication
FTA_SSL.2	Event: Re-authentication attempt to unlock a session ● user identity ● success or failure of re-authentication
FTP_ITC.1	Event: Initialization of a trusted channel ● identity of the communication partner ● protocol used to establish the channel ● success or failure of setting up the channel

Table 8: Minimum set of auditable events with event specific information

6.1.1.2 User identity association (FAU_GEN.2)

FAU_GEN.2.1 For audit events resulting from actions of identified users, the TSF shall be able to associate each auditable event with the identity of the user that caused the event.

Application Note: *The TOE maintains a "Login UID", which is inherited by every new process spawned. This allows the TOE to identify the "real" originator of an event, regardless if he has changed his real and / or effective and filesystem UID e. g. using the su or sudo commands or executing a setuid or setgid program.*

6.1.1.3 Audit review (FAU_SAR.1)

FAU_SAR.1.1	The TSF shall provide **the root user** with the capability to read **all audit information** from the audit records.
FAU_SAR.1.2	The TSF shall provide the audit records in a manner suitable for the user to interpret the information.

Application Note: *The audit records are stored in ASCII format and can therefore be read with a normal editor or pager. In addition, the TOE provides specific tools that support the interpretation of the audit trail.*

Application Note: *The audit trail is stored in a file that is readable to the users with the above mentioned capabilities only.*

6.1.1.4 Restricted audit review (FAU_SAR.2)

FAU_SAR.2.1	The TSF shall prohibit all users read access to the audit records, except those users that have been granted explicit read-access.

Application Note: *The protection of the audit records is based on the Unix permission bit settings defined by FDP_ACC.1(PSO) supported by FDP_ACF.1(PSO).*

6.1.1.5 Selective audit (FAU_SEL.1)

FAU_SEL.1.1 The TSF shall be able to select the set of events to be audited from the set of all auditable events based on the following attributes:

a) Type of audit event;
b) Subject or user identity;
c) Outcome (success or failure) of the audit event;
d) Named object identity;
e) **Access types on a particular object;**
f) **System call number;**
g) **Performing inter-field comparison rule where the specified comparison rule triggers the audit event;**
h) **arguments to system calls;**
i) **access type to file system objects (read, write, execute, change attributes);**

Application Note: *The TOE provides an application that allows specification of the audit rules which injects the rules into the kernel for enforcement. The Linux kernel auditing mechanism obtains all audit events and decides based on this rule set whether an event is forwarded to the audit daemon for storage.*

6.1.1.6 Protected audit trail storage (FAU_STG.1)

FAU_STG.1.1	The TSF shall protect the stored audit records in the audit trail from unauthorised deletion.
FAU_STG.1.2	The TSF shall be able to **prevent** unauthorised modifications to the audit records in the audit trail.

Application Note: *The protection of the audit records is based on the mechanisms explained in FAU_SAR.1.*

6.1.1.7 Action in case of possible audit data loss (FAU_STG.3)

FAU_STG.3.1 The TSF shall **notify an authorized administrator** if the audit trail exceeds **a root-user selectable, pre-defined size limit of the audit trail** or if any of the following **condition** is detected that may result in a loss of audit records *: no other condition* .

Application Note: *The term "authorized administrator" refers to the user that is notified by the auditd daemon. This daemon can be configured to notify different users in different ways. The administrator of the system must ensure that the auditd is configured to send the notification to the intended recipient.*

Application Note: *The alarm generated by the TOE can be configured to be a syslog message or the execution of an administrator-specified application. This message or action of executing the application is generated when the audit trail capacity exceeds the limit defined in the auditd.conf file.*

Application Note: *The information of the threshold limit is done in the configuration file of the auditd daemon. This file is only writable to the root user.*

6.1.1.8 Prevention of audit data loss (FAU_STG.4)

FAU_STG.4.1 The TSF shall *be able to* **ignore the audited events** and **perform one of the following administrator-defined actions:**

a) Stop all processes that attempt to generate an audit record;

b) Switch to single user mode;

c) Halt the system;

d) Notify the administrator

if the audit trail is full.

Application Note: *The SFR lists all configuration possibilities that apply to the case when the audit trail is full (i.e. the disk is full). Even though the SFR mentions the "ignoring of audit events" separate from the other options, all options should be seen as equal where the root user can select one of these options.*

6.1.1.9 Subset access control (FDP_ACC.1(PSO))

FDP_ACC.1.1 The TSF shall enforce the **Persistent Storage Object Access Control Policy** on

a) **Subjects: all subjects defined with the Security Policy Model**;

b) **Objects: all file system objects defined with the Security Policy Model**;

c) **Operations: read, write, execute (regular files), search (directories).**

6.1.1.10 Subset access control (FDP_ACC.1(TSO))

FDP_ACC.1.1 The TSF shall enforce the **Transient Storage Object Access Control Policy** on

a) **Subjects: all subjects defined with the Security Policy Model**;
b) **Objects: all IPC objects defined with the Security Policy Model**;
c) **Operations: read, receive, write, send.**

6.1.1.11 Security attribute based access control (FDP_ACF.1(PSO))

FDP_ACF.1.1 The TSF shall enforce the **Persistent Storage Object Access Control Policy** to objects based on the following:

a) Subject security attributes: file system UID, file system GID, supplemental GIDs;

b) Object security attributes: owning UID, owning GID;

c) Access control security attributes maintained for each file system object governing access to that object:

i. ACL for specific UIDs (ACL_USER),
ii. ACL for specific GIDs (ACL_GROUP),
iii. Maximum ACL for the file system object (ACL_MASK),
iv. Permission bits for the owning UID (equals to ACL_USER_OBJ when using ACLs),
v. Permission bits for the owning GID (equals to ACL_GROUP_OBJ when using ACLs),
vi. Permission bits for "world" (equals to ACL_OTHER when using ACLs),
vii. The following permission bits: read, write, execute (for files), search (for directories),
viii. The following access rights applicable to the file system object: SAVETXT (directories),

d) Access control security attributes maintained for each partition holding a file system: read-only, no-execute;

FDP_ACF.1.2 The TSF shall enforce the following rules to determine if an operation among controlled subjects and controlled objects is allowed:

A subject must have search permission for every element of the pathname and the requested access for the object. A subject has a specific type access to an object if one of the following rules hold (the order of the rules is applicable on a first-match basis):

a) The subject's filesystem UID is identical with the owning UID of the object and the requested type of access is within the permission bits defined for the owning UID (permission bits) or by ACL_USER_OBJ (ACLs); or

b) ACLs: The subject's filesystem UID is identical with the UID specified with ACL_USER of the object and the requested type of access is within the permission bits defined in ACL_USER; or

c) **The subject's filesystem GID or one of the subject's supplemental GIDs identical with the owning GID and the requested type of access is within the permission bits defined for the owning GID (permission bits), or by ACL_GROUP_OBJ when there is no ACL_MASK entry (ACLs), or by the ACL_MASK entry (ACLs); or**

d) **ACLs: The subject's filesystem GID or one of the subject's supplemental GIDs is identical with the GID specified with ACL_GROUP of the object and the requested type of access is within the permission bits defined in ACL_GROUP; or**

e) **The requested type of access is within the permission bits defined for "world" (permission bits) or by ACL_OTHER (ACLs).**

Application Note: *The permission bits and the ACLs are inherently consistent as the TOE assigns the permission bits to ACLs when ACLs are used. Without any ACLs specified for an object, the TOE only uses the permission bits. If at least one ACL is present or when the ACL management tools are applied for objects even without any ACL set, the permission bits are interpreted as outlined above: the ACL entry of ACL_USER_OBJ contains the owning UID permission bits, the ACL entry of ACL_GROUP_OBJ contains the owning GID permission bits, and the ACL entry of ACL_OTHER contains the permission bits for "world". The ACL entries of ACL_USER_OBJ, ACL_GROUP_OBJ and ACL_OTHER are only a different representation of the permission bits to users, they are not separate attributes in addition to permission bits. The explicit specification of ACL_USER_OBJ, ACL_GROUP_OBJ and ACL_OTHER in the rule set above in addition to the permission bits is only intended to aid the evaluator or reader in understanding the overall ruleset.*

Application Note: *Due to the fact that the permission bits are an inherent part of the ACLs, there is no precedence issue between permission bits and ACLs.*

FDP_ACF.1.3 The TSF shall explicitly authorise access of subjects to objects based on the following additional rules:

a) **read and directory search operations are allowed for the subject with the capability of CAP_DAC_READ_SEARCH;**

b) **write and execute operations are allowed for the subject with the capability of CAP_DAC_OVERRIDE - the execute permission is granted if the file system object object is marked with at least one executable bit in its permission settings.**

FDP_ACF.1.4 The TSF shall explicitly deny access of subjects to named objects based on the following rules:

a) **Any file system object in a file system that is mounted as read-only cannot be modified, created or removed,**

b) **A regular file, a directory and a symbolic link in a file system that is mounted as read-only cannot be written to,**

c) **A regular file in a file system that is mounted with the no-execute flag cannot be executed,**

d) **Any file system object stored in a directory marked with the SAVETXT bit cannot be modified or removed by subjects whose file system UID is not equal to the owning UID of the file system object unless the subject performing the operation possesses the CAP_FOWNER capability.**

Application Note: *The no-execute flag as well as a missing execute bit in the permission bit set or ACL for the requesting user can only be considered a convenience mechanism to prevent accidental executions of files. A missing execute permission can be circumvented using the following approaches:*

- *a binary file can be opened for reading (if the access control mechanism allows reading) with the Linux loader ld-linux.so and implicitly executed. Even without a dedicated user space loader, a user can implement the logic of the loader in an application that is marked executable to use that logic for executing any file.*
- *a script file (i.e. any ASCII file starting with a Shebang can be invoked by executing the interpreter referenced in the Shebang furnishing the ASCII file to be executed as input file.*

6.1.1.12 Security attribute based access control (FDP_ACF.1(TSO))

FDP_ACF.1.1 The TSF shall enforce the **Transient Storage Object Access Control Policy** to objects based on the following:

a) **Subject security attributes: effective UID, file system UID, effective GID, file system GID, supplemental GIDs;**

b) **Object security attributes: owning UID, owning GID;**

c) **Access control security attributes maintained for each IPC object whose name is managed with a file governing access to that object: see FDP_ACF.1(PSO);**

d) **Access control security attributes maintained for any other IPC object governing access to that object:**

   i. **Permission bits for the owning UID,**

   ii. **Permission bits for the owning GID,**

   iii. **Permission bits for "world",**

   iv. **The following permission bits: read, write, execute,**

FDP_ACF.1.2 The TSF shall enforce the following rules to determine if an operation among controlled subjects and controlled objects is allowed:

a) **IPC object whose name is managed with a file: see FDP_ACF.1(PSO);**

b) **Any other IPC object: A subject has a specific type access to an object if one of the following rules hold (the order of the rules is applicable on a first-match basis):**

   1. **The subject's effective UID is identical with the owning UID of the object and the requested type of access is within the permission bits defined for the owning UID; or**
   2. **The subject's effective GID or one of the subject's supplemental GIDs identical with the owning GID and the requested type of access is within the permission bits defined for the owning GID; or**
   3. **The requested type of access is within the permission bits defined for "world".**

FDP_ACF.1.3 The TSF shall explicitly authorise access of subjects to objects based on the following additional rules:

a) **IPC object whose name is managed with a file: see FDP_ACF.1(PSO);**

b) **Any other IPC object:**

1. **read, write, send and receive operations are allowed for the subject with the capability of CAP_IPC_OWNER.**

FDP_ACF.1.4 The TSF shall explicitly deny access of subjects to named objects based on the following rules:

- **a) IPC object whose name is managed with a file: see FDP_ACF.1(PSO);**
- **b) Any other IPC object: none.**

6.1.1.13 Complete information flow control (FDP_IFC.1)

FDP_IFC.1.1 The TSF shall enforce the Network Information Flow Control Policy on

- a) Originating entities:
  - i. unauthenticated external IT entities that send network data to a network interface of the TOE;
  - ii. subjects within the TOE that send network data to unauthenticated external IT entities via a network interface of the TOE;
- b) Information:
  - i. Network data received by the TOE from an external IT entity;
  - ii. Network data provided to the TOE by a subject executing on the TOE intended to be sent to an external IT entity via a network interface controlled by the TOE;
  - iii. **No other information;**
- c) Operations:
  - i. Receiving network data from an unauthenticated external IT entity;
  - ii. Sending network data to an unauthenticated IT entity by a subject within the TOE;

6.1.1.14 Simple security attributes (FDP_IFF.1)

FDP_IFF.1.1 The TSF shall enforce the Network Information Flow Control Policy based on the following types of subject and information security attributes:

- a) ~~Object~~*Information* security attribute: the logical or physical network interface through which the network data from an external IT entity entered the TOE or is intended to be sent out;
- b) **TCP/IP information security attributes:**
  - **i. Source and destination IP address,**
  - **ii. Source and destination TCP port number,**
  - **iii. Source and destination UDP port number,**
  - **iv. Network protocol of TCP, UDP, ICMP**
  - **v. TCP header flags of SYN, ACK, FIN, RST, URG, PSH, TCP sequence numbers**

Application Note: *The refinement is applied due to an obvious error in the OSPP.*

FDP_IFF.1.2 The TSF shall permit an information flow between a controlled subject and controlled information via a controlled operation if the following rules hold: for both receiving network data from an external IT entity and sending network data by a subject within the TOE to an external IT entity:

a) if the set of rules defined in accordance with the security attributes defined in FDP_IFF.1.3 define that the network data is discarded the network data shall not be delivered by the TOE to the intended recipient;

b) if the set of rules defined in accordance with the security attributes defined in FDP_IFF.1.3 define that the network data is to be delivered unaltered the network data shall be delivered unaltered by the TOE to the intended recipient;

c) if the set of rules defined in accordance with the security attributes defined in FDP_IFF.1.3 define another action to be taken than discarding the network data or delivering the data unaltered to the intended recipient, the TOE shall perform this action.

FDP_IFF.1.3 The TSF shall enforce the following rules consisting of an identification when the rule fires and an action to be taken when the rule fires:

a) Information security attribute matching based on the following security attributes:

   1. IP header information,

   2. UDP header information,

   3. TCP header information,

   4. ICMP type and code,

   5. incoming network interface,

   6. outgoing network interface

b) **Matching based on the state of a TCP connection, Statistical analysis matching**;

Performing one or more of the following actions with identified network data:

a) Discard the network data **without any further processing, with sending a notification to the sender**;

b) Allow the network data to be processed unaltered by the TOE according to the routing information maintained by the TOE;

c) **No other actions**.

FDP_IFF.1.4 The TSF shall explicitly authorise an information flow based on the following rules: **If the network data is not matched by the rule set and the default rule of the packet filter is ACCEPT then the data is forwarded unaltered based on the normal operation of the host system's networking stack** .

FDP_IFF.1.5 The TSF shall explicitly deny an information flow based on the following rules: **If the network data is not matched by the rule set, one of the following default rules applies:**

a) DROP: the data is discarded.

Application Note: *The default rule is configurable where exactly one of the above mentioned default rules can be selected at any given time.*

Application Note: *The SFRs FDP_IFF.1 defines different rule sets implemented by the TOE covering the FDP_IFF.1 SFR from the OSPP base.*

6.1.1.15 Full residual information protection (FDP_RIP.2)

FDP_RIP.2.1 The TSF shall ensure that any previous information content of a resource is made unavailable upon the **allocation of the resource to** all objects, subjects or subject/object related TSF data before the resource is assigned or made available to another subject or user.

6.1.1.16 Authentication failure handling (FIA_AFL.1)

FIA_AFL.1.1 The TSF shall detect when an administrator-configurable positive integer within a range of acceptable values of unsuccessful authentication attempts for the authentication method password based authentication **no other method** occur related to **consecutive unsuccessful authentication attempts**.

FIA_AFL.1.2 When the defined number of unsuccessful authentication attempts has been met, the TSF shall

- **a) For all administrator accounts, "disable" the account for an authorized administrator configurable time period such that there can be no more than ten attempts per minute.**
- **b) For all other accounts, disable the user logon account until it is re-enabled by the authorized administrator.**
- **c) For all disabled accounts, any response to an authentication attempt given to the user shall not be based on the result of that authentication attempt.**

6.1.1.17 User attribute definition (FIA_ATD.1)

FIA_ATD.1.1 The TSF shall maintain the following list of security attributes belonging to individual human users:

- a) User identifier;
- b) Group memberships;
- c) User password;
- d) Security roles;
- e) **Software token verification data;**

Application Note: *Please see the application note for FIA_UAU.5 for a list of token-based authentication mechanisms and their associated tokens.*

6.1.1.18 Timing of authentication (FIA_UAU.1(RITE))

FIA_UAU.1.1 The TSF shall allow

- a) the information flow covered by the Network Information Flow Control Policy;
- b) **SSH log-in: obtaining the list of allowed authentication methods;**
- c) **Establishing a cryptographically secured network connection;**

on behalf of the remote IT entity to be performed before the remote IT entity is authenticated.

FIA_UAU.1.2 The TSF shall require each user to be successfully authenticated before allowing any other TSF-mediated actions on behalf of that remote IT entity.

6.1.1.19 Timing of authentication (FIA_UAU.1(HU))

FIA_UAU.1.1 The TSF shall allow

a) **Local console log-in: banner information;**

on behalf of the user to be performed before the user is authenticated.

FIA_UAU.1.2 The TSF shall require each user to be successfully authenticated before allowing any other TSF-mediated actions on behalf of that user.

6.1.1.20 Multiple authentication mechanisms (FIA_UAU.5)

FIA_UAU.5.1 The TSF shall provide the following authentication mechanisms:

a) Authentication based on username and password (for human users);

b) **Authentication based on SSH keys**

c) **Authentication based on remote authentication provider**

to support user authentication.

Application Note: *The TOE is able to maintain the following types of software tokens and their verification data:*

- *SSH user keys: The TOE as server part is able to store the public part of the SSH user key for the user account the user wants to access. When the TOE acts as an SSH client, the TOE is able to store the private part of the SSH user key for the requesting user.*
- *Kerberos token: The TOE stores the Kerberos token retrieved from a Kerberos Ticket Granting Server.*

FIA_UAU.5.2 The TSF shall authenticate any user's claimed identity according to the following rules:

a) Authentication based on username and password is performed for TOE-originated requests and with credentials stored by the TSF by default unless another authentication method defined for human users in FIA_UAU.5.1 b is selected;

b) Users with expired passwords are **required to create a new password after correctly entering the expired password**

c) **For SSH, both, the password-based and key-based authentication methods can be enabled at the same time. In this case, the key-based authentication method is tried before the password-based authentication. If the key-based authentication succeeds, the user is authenticated. If the key-based authentication fails, the password-based authentication is applied. If the password-based authentication fails, the user login request is denied.**

d) **For username and password based authentication, the order whether the remote authentication provider or the local database is accessed is configurable. If the authentication at either the locally store credentials or at the remote authentication provider succeeds, the authenticating user is granted access.**

6.1.1.21 Protected authentication feedback (FIA_UAU.7)

FIA_UAU.7.1 The TSF shall provide only obscured feedback to the user while the authentication is in progress.

6.1.1.22 Timing of identification (FIA_UID.1)

FIA_UID.1.1 The TSF shall allow

a) Establishing a cryptographically secured network connection;

b) Console log-in: banner information;

c) SSH log-in: obtaining the list of allowed authentication methods;

on behalf of the user to be performed before the user is identified.

FIA_UID.1.2 The TSF shall require each user to be successfully identified before allowing any other TSF-mediated actions on behalf of that user.

6.1.1.23 User-subject binding (FIA_USB.1)

FIA_USB.1.1 The TSF shall associate the following security attributes with subjects acting on the behalf of that human user:

a) The user identity;

b) **The user security attributes that are used to enforce the Persistent Storage Object Access Control Policy;**

c) **The user security attributes that are used to enforce the Transient Storage Object Access Control Policy;**

d) **The SSH private key that can be used for subsequent identification and authentication with the TSF or other remote IT systems;**

e) **Active roles;**

f) **Active groups;**

FIA_USB.1.2 The TSF shall enforce the following rules on the initial association of security attributes with subjects acting on the behalf of users:

a) Upon successful identification and authentication, the login UID, the real UID, the filesystem UID and the effective UID shall be those specified in the user entry for the user that has authenticated successfully;

b) Upon successful identification and authentication, the real GID, the filesystem GID and the effective GID shall be those specified via the primary group membership attribute in the user entry;

c) Upon successful identification and authentication, the supplemental GIDs shall be those specified via the supplemental group membership assignment for the user entry;

Application Note: *The various subject UIDs are all derived from the same numeric UID per user entry stored in the /etc/passwd file.*

Application Note: *The various subject GIDs except the supplemental GIDs are all derived from the same numeric GID per user entry stored in the /etc/passwd file.*

Application Note: *The subject's supplemental GIDs are derived from the username to group name mappings in the /etc/group file. As the TOE only maintains numeric IDs for subjects, the username and the group names need to be converted before instantiating the subject. The username to UID mapping is provided in /etc/passwd and the group name to GID mapping is provided in /etc/group.*

FIA_USB.1.3 The TSF shall enforce the following rules governing changes to the user security attributes associated with subjects acting on the behalf of users:

a) **The effective and filesystem UID of a subject can be changed by the use of an executable with the SETUID bit set. In this case the program is executed with the effective and filesystem UID of the owning UID of the file storing the program. These newly set effective and filesystem UIDs are used for the DAC permission validation. The real and login UID remain unchanged.**

b) **The effective and filesystem GID of a subject can be changed by the use of an executable with the SETGID bit set. In this case the program is executed with the effective and filesystem GID of the owning GID of the file storing the program. These newly set effective and filesystem GIDs are used for the DAC permission validation. The real GID remains unchanged.**

c) **The real, effective and filesystem UID of a subject can be changed by the use of the set*uid system call family for the calling application. These system calls are restricted to processes possessing the CAP_SETUID capability.**

d) **The real, effective and filesystem GID of a subject can be changed by the use of the set*gid system call family for the calling application. These system calls are restricted to processes possessing the CAP_SETUID capability.**

e) **The set of supplemental GIDs of a subject can be changed by the use of the setgroups system call for the calling application. These system calls are restricted to processes possessing the CAP_SETUID capability.**

f) **The set of effective and inheritable capabilities of a subject can be changed by the use of an executable with activated file capabilities. In this case the program obtains the following capabilities when invoking the file with execve:**

   1. **the process' the effective capability set gains the capabilities defined by the permitted file capabilities set;**
   2. **the process' inheritable capability set is ANDed with the inheritable file capability set to form the new process' inheritable capability set which defines the capability set that will be retained after an execve system call.**

Application Note: *The applications "su" and "sudo" allow the calling user to change the filesystem and effective UID either to root or to other users provided the authentication to "su" or "sudo" was successful. Both application uses the SETUID bit with the owning UID of root as well as the set*uid system calls to change to other UIDs before spawning a new shell or the given command. As both applications rest on the above mentioned mechanisms, it is not listed as a separate mechanism to modify the calling user's UIDs.*

Application Note: *The login UID is set by the PAM modules by inserting the intended UID into the /proc/<PID>/loginuid file. This file can be written to only by subjects executing with the effective UID of zero (root) and only for the calling process' own loginuid file. However, there is no application except the PAM modules which access that proc file which implies that the login UID remains unchanged after login when operating the TOE. Authorized administrators are not intended to access that proc file.*

6.1.1.24 Public key based authentication (FIA_PK_EXT.1)

FIA_PK_EXT.1.1 The TSF shall use **public key cryptography** as defined by **the SSH protocol specification given in [RFC4352]☝ for SSH public key authentication** to support authentication for **SSH** connections.

FIA_PK_EXT.1.2 The TSF shall store and protect certificates and/or public keys from unauthorized deletion and modification.

6.1.1.25 Reliable time stamps (FPT_STM.1)

FPT_STM.1.1 The TSF shall be able to provide reliable time stamps.

6.1.1.26 TSF-initiated session locking (FTA_SSL.1)

FTA_SSL.1.1 The TSF shall lock an interactive session to a human user maintained by the TSF after **an administrator-configurable time interval of user inactivity** by:

a) clearing or overwriting TSF controlled display devices, making the current contents unreadable;
b) disabling any activity of the user's TSF controlled access/TSF controlled display devices other than unlocking the session.

FTA_SSL.1.2 The TSF shall require the following events to occur prior to unlocking the session:

a) Successful re-authentication with the credentials of the user owning the session using **password based authentication**;
b) **No other events** .

6.1.1.27 User-initiated locking (FTA_SSL.2)

FTA_SSL.2.1 The TSF shall allow user-initiated locking of the user's own interactive session maintained by the TSF, by:

a) clearing or overwriting TSF controlled display devices, making the current contents unreadable;
b) disabling any activity of the user's TSF controlled data access/TSF controlled display devices other than unlocking the session.

FTA_SSL.2.2 The TSF shall require the following events to occur prior to unlocking the session:

a) Successful re-authentication with the credentials of the user owning the session using **password based authentication**;
b) **No other events** .

6.1.1.28 Inter-TSF trusted channel (FTP_ITC.1)

FTP_ITC.1.1 The TSF shall provide a communication channel between itself and another trusted IT product that is logically distinct from other communication channels and provides assured identification of its end points and protection of the channel data from modification and disclosure using the following mechanisms: Cryptographically-protected communication channel using

i. **SSH protocol version 2 as defined in RFCs 4251, 4252, 4253, and 4254 with a combination of the following cipher suites defined there:**

1. **3DES-CBC, AES256-CBC, AES128-CBC, AES192-CBC, AEAD_AES_128_GCM (as defined in RFC 5647), AEAD_AES_256_GCM (as defined in RFC 5647) for encryption;**
2. **HMAC_SHA1, HMAC-SHA1-96 for integrity;**
3. **DIFFIE-HELLMAN-GROUP14-SHA1, DIFFIE-HELLMAN-GROUP1-SHA1 for key exchange;**
4. **SSH-DSS, SSH-RSA, no other public key algorithms for public key encryption;**

FTP_ITC.1.2 The TSF shall permit **the TSF, another trusted IT product** to initiate communication via the trusted channel.

FTP_ITC.1.3 The TSF shall initiate communication via the trusted channel for all security functions specified in the ST that interact with remote trusted IT systems and **no other conditions or functions**.

Application Note: *The SSH protocol implements a bi-directional authentication mechanism as follows:*

- *Server-side authentication: the user identification and authentication via user name and password / SSH user key allows the server to authenticate the client.*
- *Client-side authentication: the SSH host key verification performed by the SSH client during each connection attempt allows the client to authenticate the server.*

6.1.2 Management related functionality

6.1.2.1 Management of security functions behaviour (FMT_MOF.1)

FMT_MOF.1.1 The TSF shall restrict the ability to modify the behaviour of the functions password based user authentication to **the root user** by allowing those users to specify rules for acceptable passwords that:

a) allow for uppercase characters, lowercase characters, digits, and special characters to be used in passwords
b) define a minimum password length of 8 characters or more (at least up to 15 characters)
c) define that passwords must have at least one digit and one special character
d) reject passwords used by the same user before up to a history of at least 6 passwords

6.1.2.2 Management of object security attributes (FMT_MSA.1(PSO))

FMT_MSA.1.1 The TSF shall enforce the **Persistent Storage Object Access Control Policy** to restrict the ability to modify and **change_default** the security attributes of the objects covered by the SFP to the owner of the object and **root user** .

6.1.2.3 Management of object security attributes (FMT_MSA.1(TSO))

FMT_MSA.1.1 The TSF shall enforce the **Transient Storage Object Access Control Policy** to restrict the ability to modify the security attributes of the objects covered by the SFP to the owner of the object and **users with processes granted the CAP_FOWNER capability**.

6.1.2.4 Static attribute initialisation (FMT_MSA.3(DAC/PSO))

FMT_MSA.3.1 The TSF shall enforce the **Persistent Storage Object Access Control Policy** to provide restrictive default values for security attributes that are used to enforce the SFP.

FMT_MSA.3.2 The TSF shall allow the

a) root user for a global setting applied during logon;

b) each user for a setting applicable to his processes;

c) users with write permissions to a directory for setting default ACLs

to specify alternative initial values to override the default values when an object or information is created.

Application Note: *The global default value for permission bits is specified with the umask value which specifies the permission bits for newly created objects. This value has an initial setting of 022 or the value specified in /etc/login.defs. Only the root user can manage that initial value as this file is writable to root only. Users can change their umask value at any time using the umask(2) system call. For ACLs, the default ACL is provided for for the root directory which, in case of absence of a default ACL entry is consistent with the umask.*

6.1.2.5 Static attribute initialisation (FMT_MSA.3(DAC/TSO))

FMT_MSA.3.1 The TSF shall enforce the **Transient Storage Object Access Control Policy** to provide restrictive default values for security attributes that are used to enforce the SFP.

FMT_MSA.3.2 The TSF shall allow the

a) root user for a global setting applied during logon;

b) each user for a setting applicable to his processes

to specify alternative initial values to override the default values when an object or information is created.

Application Note: *The global default value for permission bits is specified with the umask value which specifies the permission bits for newly created objects. This value has an initial setting of 022 or the value specified in /etc/login.defs. Only the root user can manage that initial value as this file is writable to root only. Users can change their umask value at any time using the umask(2) system call.*

6.1.2.6 Static attribute initialisation (FMT_MSA.3(NI))

FMT_MSA.3.1 The TSF shall enforce the Network Information Flow Control Policy to provide **permissive** default values for security attributes that are used to enforce the SFP.

FMT_MSA.3.2 The TSF shall allow the **root user** to specify alternative initial values to override the default values when an object or information is created.

Application Note: *The default value specified in this SFR applies to the default for the packet filter after boot. The administrator can configure alternative default values as outlined in FDP_IFF.1.*

Application Note: *The iptables command uses a netlink interfact to the kernel which requires that the caller possesses the CAP_NET_ADMIN capability.*

6.1.2.7 Security attribute value inheritance (FMT_MSA.4)

FMT_MSA.4.1 The TSF shall use the following rules to set the value of security attributes for objects covered by an access control policy:

a) The newly created object's owning UID is set to the effective UID of the calling subject;

b) The newly created object's owning GID is set to the effective GID of the calling subject with the following exception for file system objects: if the parent directory holding the newly created file system object is marked with the SETGID permission bit, the owning GID of the newly created file system object is set to the owning GID of the parent directory;

c) The newly created object's permission bits are derived from the calling subject's umask value by masking out the umask bits from the permission bit set granting full access;

d) The newly created object's ACLs are derived from the default ACL specified for the parent directory the newly created file system object is stored in, if existant. Otherwise, no ACL is set.

.

6.1.2.8 Management of TSF data (FMT_MTD.1(AE))

FMT_MTD.1.1 The TSF shall restrict the ability to query, modify the set of audited events to **processes with the capability CAP_AUDIT_CONTROL**.

Application Note: *This SFR applies to FAU_SEL.1.*

Application Note: *Using the audit tools which in turn use the netlink interface, an administrator can configure the audit rules.*

6.1.2.9 Management of TSF data (FMT_MTD.1(AS))

FMT_MTD.1.1 The TSF shall restrict the ability to clear **delete, configure the storage location** the audit storage to **the root user**.

Application Note: *This SFR applies to FAU_STG.1 where the directory used for storing the audit trail is configured.*

Application Note: *The configuration of these parameters is performed with the configuration file /etc/auditd/auditd.conf which is writable to the root user only.*

6.1.2.10 Management of TSF data (FMT_MTD.1(AT))

FMT_MTD.1.1 The TSF shall restrict the ability to modify **add, delete** the

a) threshold of the audit trail when an action is performed;

b) action when the threshold is reached

to **the root user**.

Application Note: *This SFR applies to FAU_STG.3.*

Application Note: *The configuration of these parameters is performed with the configuration file /etc/auditd/auditd.conf which is writable to the root user only.*

6.1.2.11 Management of TSF data (FMT_MTD.1(AF))

FMT_MTD.1.1 The TSF shall restrict the ability to modify **add, delete** the actions to be taken in case of audit storage failure to **the root user**.

Application Note: *This SFR applies to FAU_STG.4.*

Application Note: *The configuration of these parameters is performed with the configuration file /etc/auditd/auditd.conf which is writable to the root user only.*

6.1.2.12 Management of TSF data (FMT_MTD.1(CM))

FMT_MTD.1.1 The TSF shall restrict the ability to import, enable, disable the digital certificates or public keys for remote entity authentication **no other security function** to

a) the account owner and the user with the CAP_DAC_OVERRIDE for SSH

.

Application Note: *This SFR applies to FTP_ITC.1.*

6.1.2.13 Management of TSF data (FMT_MTD.1(NI))

FMT_MTD.1.1 The TSF shall restrict the ability to define, query, modify, delete **change_default** the security attributes for the rules governing the

a) identification of and matching of network data;

b) actions performed on the identified network data;

to **users with processes granted the CAP_NET_ADMIN capability**.

Application Note: *This SFR applies to FDP_IFF.1.*

Application Note: *The iptables command use a netlink interface to the kernel which requires that the caller possesses the CAP_NET_ADMIN capability.*

6.1.2.14 Management of TSF data (FMT_MTD.1(IAT))

FMT_MTD.1.1 The TSF shall restrict the ability to modify the threshold for unsuccessful authentication attempts to **the root user**.

Application Note: *This SFR applies to FIA_AFL.1.*

Application Note: *The configuration of these parameters is performed with the PAM configuration files which are writable to the root user only.*

6.1.2.15 Management of TSF data (FMT_MTD.1(IAF))

FMT_MTD.1.1 The TSF shall restrict the ability to re-enable the authentication to the account subject to authentication failure to **the root user**.

Application Note: *This SFR applies to FIA_AFL.1.*

Application Note: *The account locking information is stored in the directory /var/log/faillock. Using the pam_faillock application which modifies this file, the account can be unlocked. The DAC permissions of that file ensure that only the root user can write to it.*

6.1.2.16 Management of TSF data (FMT_MTD.1(IAU))

FMT_MTD.1.1 The TSF shall restrict the ability to initialize, modify, delete the user security attributes *stored in local databases* to

a) the root user,

b) users authorized to modify their own authentication data

.

Application Note: *This SFR applies to FIA_ATD.1, FIA_UAU.1(HU), FIA_UAU.1(RITE), and FIA_UID.1.*

Application Note: *The configuration of these parameters is performed with the configuration files /etc/passwd and /etc/shadow which are writable to the root user only. The TOE also supports remote authentication data stores such as LDAP or Kerberos. In this case, the TOE does not have the ability to protect these databases. This is ensured by the assumption of A.PEER.MGT.*

6.1.2.17 Revocation (FMT_REV.1(OBJ))

FMT_REV.1.1 The TSF shall restrict the ability to revoke object security attributes defined by SFPs associated with the corresponding object under the control of the TSF to

a) DAC permissions: owners of the object and authorized administrator;

b) Other security attributes: authorized administrator.

Application Note: *The privileges that constitute an authorized administrator are defined in the above mentioned FMT_* SFRs which specify the privileges needed to modify object security attributes. The same privileges are required to revoke these security attributes.*

FMT_REV.1.2 The TSF shall enforce the following rules:

a) The access rights associated with an object shall be enforced when an access check is made;

b) **no specification of other revocation rules**.

Application Note: *Revocation of security attributes for named objects imply the revocation of access granted to users other than the owner of the object. Note that the DAC ownership management (which can be also considered as a form of access revocation) is specified in FMT_MSA.1(PSO).*

6.1.2.18 Revocation (FMT_REV.1(USR))

FMT_REV.1.1 The TSF shall restrict the ability to revoke user security attributes defined by the SFP associated with the corresponding user under the control of the TSF to **authorized administrators** .

Application Note: *The privileges that constitute an authorized administrator are defined in the above mentioned FMT_* SFRs which specify the privileges needed to modify object security attributes. The same privileges are required to revoke these security attributes.*

FMT_REV.1.2 The TSF shall enforce the following rules:

a) The enforcement of the revocation of security-relevant authorizations with the next user-subject binding process during the next authentication of the user;

b) **No other rules**

Application Note: *The changes are enforced for a new session when the user affected by the change initiates that new session.*

6.1.2.19 Remote Management Capabilities (FMT_SMF_RMT.1)

FMT_SMF_RMT.1.1 The TSF shall allow management functions also to be performed from a remote IT entity using a trusted channel established in accordance with the requirements stated in FTP_ITC.1.

6.1.2.20 Security management roles (FMT_SMR.1)

FMT_SMR.1.1 The TSF shall maintain the roles:

a) authorized administrator;

b) regular user;

c) **no other roles.**

FMT_SMR.1.2 The TSF shall be able to associate users with roles.

Application Note: *Administrative actions can only be performed when the calling subject possesses the above mentioned capabilities which in the TOE configuration is only provided to processes executing with the effective UID or file system UID of zero (also called the root user). As the account for the root user is disabled for direct logon, authorized administrators are defined as users who are assigned to the "wheel" group. This group allows the use of the "su" application which is the only way to assume the root user capabilities. In addition, the "sudo" application allows granting users the privilege to execute commands with a different user ID, including the root user.*

6.2 Security Functional Requirements Rationale

6.2.1 Coverage

The following table provides a mapping of SFR to the security objectives, showing that each security functional requirement addresses at least one security objective.

Security functional requirements	**Objectives**
FAU_GEN.1	O.AUDITING
FAU_GEN.2	O.AUDITING
FAU_SAR.1	O.AUDITING
FAU_SAR.2	O.AUDITING
FAU_SEL.1	O.AUDITING
FAU_STG.1	O.AUDITING
FAU_STG.3	O.AUDITING
FAU_STG.4	O.AUDITING
FDP_ACC.1(PSO)	O.DISCRETIONARY.ACCESS
FDP_ACC.1(TSO)	O.SUBJECT.COM

Security functional requirements	**Objectives**
FDP_ACF.1(PSO)	O.DISCRETIONARY.ACCESS
FDP_ACF.1(TSO)	O.SUBJECT.COM
FDP_IFC.1	O.NETWORK.FLOW
FDP_IFF.1	O.NETWORK.FLOW
FDP_RIP.2	O.AUDITING, O.DISCRETIONARY.ACCESS, O.I&A, O.NETWORK.FLOW, O.SUBJECT.COM
FIA_AFL.1	O.I&A
FIA_ATD.1	O.I&A
FIA_UAU.1(RITE)	O.I&A, O.NETWORK.FLOW
FIA_UAU.1(HU)	O.I&A
FIA_UAU.5	O.I&A
FIA_UAU.7	O.I&A
FIA_UID.1	O.I&A
FIA_USB.1	O.I&A
FIA_PK_EXT.1	O.TRUSTED_CHANNEL
FPT_STM.1	O.AUDITING
FTA_SSL.1	O.UNATTENDED_SESSION
FTA_SSL.2	O.UNATTENDED_SESSION
FTP_ITC.1	O.TRUSTED_CHANNEL
FMT_MOF.1	O.I&A, O.MANAGE
FMT_MSA.1(PSO)	O.MANAGE
FMT_MSA.1(TSO)	O.MANAGE
FMT_MSA.3(DAC/PSO)	O.MANAGE
FMT_MSA.3(DAC/TSO)	O.MANAGE
FMT_MSA.3(NI)	O.MANAGE
FMT_MSA.4	O.MANAGE
FMT_MTD.1(AE)	O.MANAGE

Security functional requirements	Objectives
FMT_MTD.1(AS)	O.MANAGE
FMT_MTD.1(AT)	O.MANAGE
FMT_MTD.1(AF)	O.MANAGE
FMT_MTD.1(CM)	O.MANAGE
FMT_MTD.1(NI)	O.MANAGE
FMT_MTD.1(IAT)	O.MANAGE
FMT_MTD.1(IAF)	O.MANAGE
FMT_MTD.1(IAU)	O.MANAGE
FMT_REV.1(OBJ)	O.MANAGE
FMT_REV.1(USR)	O.MANAGE
FMT_SMF_RMT.1	O.MANAGE
FMT_SMR.1	O.MANAGE

Table 9: Mapping of security functional requirements to security objectives

6.2.2 Sufficiency

The following rationale provides justification for each security objective for the TOE, showing that the security functional requirements are suitable to meet and achieve the security objectives.

Security objectives	Rationale
O.AUDITING	The events to be audited are defined in [FAU_GEN.1] and are associated with the identity of the user that caused the event [FAU_GEN.2]. Authorized users are provided the capability to read the audit records [FAU_SAR.1], while all other users are denied access to the audit records [FAU_SAR.2]. The authorized user must have the capability to specify which audit records are generated [FAU_SEL.1]. The TOE prevents the audit log from being modified or deleted [FAU_STG.1] and ensures that the audit log is not lost due to resource shortage [FAU_STG.3, FAU_STG.4]. To support auditing, the TOE is able to maintain proper time stamps [FPT_STM.1]. The protection of reused resources ensures that no data leaks from other protected sources [FDP_RIP.2].
O.DISCRETIONARY.ACCESS	The TSF must control access to resources based on the identity of users that are allowed to specify which resources they want to access for storing their data. The access control policy must have a defined scope of control [FDP_ACC.1(PSO)]. The rules for the access control policy are defined [FDP_ACF.1(PSO)].

Security objectives	Rationale
	The protection of reused resources ensures that no data leaks from other protected sources [FDP_RIP.2].
O.NETWORK.FLOW	The network information flow control mechanism controls the information flowing between different entities [FDP_IFC.1]. The TOE implements a rule-set governing the information flow [FDP_IFF.1]. Information flow control is enforced for unauthenticated remote IT entity, allowing authenticated remote IT entity to be excluded from the rules of the network information flow control policy (FIA_UAU.1(RITE)). The protection of reused resources ensures that no data leaks from other protected sources [FDP_RIP.2].
O.SUBJECT.COM	The TSF must control the exchange of data using transient storage objects between subjects based on the identity of users. The access control policy must have a defined scope of control [FDP_ACC.1(TSO)]. The rules for the access control policy are defined [FDP_ACF.1(TSO)]. The protection of reused resources ensures that no data leaks from other protected sources [FDP_RIP.2].
O.I&A	The TSF must ensure that only authorized users gain access to the TOE and its resources. Users authorized to access the TOE must use an identification and authentication process [FIA_UID.1, FIA_UAU.1(HU), FIA_UAU.1(RITE)]. Multiple I&A mechanisms are allowed as specified in [FIA_UAU.5]. To ensure authorized access to the TOE, authentication data is protected [FIA_ATD.1, FIA_UAU.7]. Proper authorization for subjects acting on behalf of users is also ensured [FIA_USB.1]. To support the strength of authentication methods, the TOE is capable of identifying and reacting to unsuccessful authentication attempts [FIA_AFL.1] and define password rules [FMT_MOF.1]. The protection of reused resources ensures that no data leaks from other protected sources [FDP_RIP.2] are present.
O.MANAGE	The TOE provides management interfaces for: ● the access control policies [FMT_MSA.1(PSO), FMT_MSA.1(TSO), FMT_MSA.3(DAC/PSO), FMT_MSA.3(DAC/TSO)]; ● the information flow control policy [FMT_MSA.3(NI), FMT_MTD.1(NI)]; ● the auditing aspects [FMT_MTD.1(AE), FMT_MTD.1(AS), FMT_MTD.1(AT), FMT_MTD.1(AF)]; ● digital certificates [FMT_MTD.1(CM}]; ● the identification and authentication aspects [FMT_MTD.1(IAT), FMT_MTD.1(IAF), FMT_MTD.1(IAU)]. Persistently stored user data is stored either in hierarchical or relational fashion, which implies an inheritance of security attributes from parent object [FMT_MSA.4]. The rights management for the different management aspects is defined with [FMT_SMR.1].

Security objectives	Rationale
	The management interfaces for the revocation of user and object attributes is provided with [FMT_REV.1(OBJ) and FMT_REV.1(USR)]. Management of password rules is defined in [FMT_MOF.1]. Remote management capabilities need to be provided as defined in [FMT_SMF_RMT.1].
O.TRUSTED_CHANNEL	The TOE provides a trusted channel protecting communication between a remote trusted IT system and itself [FTP_ITC.1]. Digital certificates must be used for remote entity authentication [FIA_PK_EXT.1].
O.UNATTENDED_SESSION	User-initiated and TSF-initiated session locking [FTA_SSL.1, FTA_SSL.2] protect the authenticated user's session.

Table 10: Security objectives for the TOE rationale

6.2.3 Security requirements dependency analysis

The following table demonstrates the dependencies of SFRs modeled in CC Part 2 and how the SFRs for the TOE resolve those dependencies.

Security functional requirement	Dependencies	Resolution
FAU_GEN.1	FPT_STM.1	FPT_STM.1
FAU_GEN.2	FAU_GEN.1	FAU_GEN.1
	FIA_UID.1	FIA_UID.1
FAU_SAR.1	FAU_GEN.1	FAU_GEN.1
FAU_SAR.2	FAU_SAR.1	FAU_SAR.1
FAU_SEL.1	FAU_GEN.1	FAU_GEN.1
	FMT_MTD.1	FMT_MTD.1(AE)
FAU_STG.1	FAU_GEN.1	FAU_GEN.1
FAU_STG.3	FAU_STG.1	FAU_STG.1
FAU_STG.4	FAU_STG.1	FAU_STG.1
FDP_ACC.1(PSO)	FDP_ACF.1	FDP_ACF.1(PSO)
FDP_ACC.1(TSO)	FDP_ACF.1	FDP_ACF.1(TSO)
FDP_ACF.1(PSO)	FDP_ACC.1	FDP_ACC.1(PSO)
	FMT_MSA.3	FMT_MSA.3(DAC/PSO)

Security functional requirement	Dependencies	Resolution
FDP_ACF.1(TSO)	FDP_ACC.1	FDP_ACC.1(TSO)
	FMT_MSA.3	FMT_MSA.3(DAC/TSO)
FDP_IFC.1	FDP_IFF.1	FDP_IFF.1
FDP_IFF.1	FDP_IFC.1	FDP_IFC.1
	FMT_MSA.3	FMT_MSA.3(NI)
FDP_RIP.2	No dependencies	
FIA_AFL.1	FIA_UAU.1	FIA_UAU.1(HU)
FIA_ATD.1	No dependencies	
FIA_UAU.1(RITE)	FIA_UID.1	FIA_UID.1
FIA_UAU.1(HU)	FIA_UID.1	FIA_UID.1
FIA_UAU.5	No dependencies	
FIA_UAU.7	FIA_UAU.1	FIA_UAU.1(HU)
FIA_UID.1	No dependencies	
FIA_USB.1	FIA_ATD.1	FIA_ATD.1
FIA_PK_EXT.1	FMT_MTD.1	FMT_MTD.1(CM)
FPT_STM.1	No dependencies	
FTA_SSL.1	FIA_UAU.1	FIA_UAU.1(HU)
FTA_SSL.2	FIA_UAU.1	FIA_UAU.1(HU)
FTP_ITC.1	No dependencies	
FMT_MOF.1	FMT_SMR.1	FMT_SMR.1
	FMT_SMF.1	See OSPP rationale.
FMT_MSA.1(PSO)	[FDP_ACC.1 or FDP_IFC.1]	FDP_ACC.1(PSO)
	FMT_SMR.1	FMT_SMR.1
	FMT_SMF.1	See OSPP rationale.
FMT_MSA.1(TSO)	[FDP_ACC.1 or FDP_IFC.1]	FDP_ACC.1(PSO)
	FMT_SMR.1	FMT_SMR.1
	FMT_SMF.1	See OSPP rationale.

Security functional requirement	Dependencies	Resolution
FMT_MSA.3(DAC/PSO)	FMT_MSA.1	FMT_MSA.1(PSO)
	FMT_SMR.1	FMT_SMR.1
FMT_MSA.3(DAC/TSO)	FMT_MSA.1	FMT_MSA.1(TSO)
	FMT_SMR.1	FMT_SMR.1
FMT_MSA.3(NI)	FMT_MSA.1	See OSPP rationale.
	FMT_SMR.1	FMT_SMR.1
FMT_MSA.4	[FDP_ACC.1 or FDP_IFC.1]	FDP_ACC.1(PSO)
FMT_MTD.1(AE)	FMT_SMR.1	FMT_SMR.1
	FMT_SMF.1	See OSPP rationale.
FMT_MTD.1(AS)	FMT_SMR.1	FMT_SMR.1
	FMT_SMF.1	See OSPP rationale.
FMT_MTD.1(AT)	FMT_SMR.1	FMT_SMR.1
	FMT_SMF.1	See OSPP rationale.
FMT_MTD.1(AF)	FMT_SMR.1	FMT_SMR.1
	FMT_SMF.1	See OSPP rationale.
FMT_MTD.1(CM)	FMT_SMR.1	FMT_SMR.1
	FMT_SMF.1	See OSPP rationale.
FMT_MTD.1(NI)	FMT_SMR.1	FMT_SMR.1
	FMT_SMF.1	See OSPP rationale.
FMT_MTD.1(IAT)	FMT_SMR.1	FMT_SMR.1
	FMT_SMF.1	See OSPP rationale.
FMT_MTD.1(IAF)	FMT_SMR.1	FMT_SMR.1
	FMT_SMF.1	See OSPP rationale.
FMT_MTD.1(IAU)	FMT_SMR.1	FMT_SMR.1
	FMT_SMF.1	See OSPP rationale.
FMT_REV.1(OBJ)	FMT_SMR.1	FMT_SMR.1
FMT_REV.1(USR)	FMT_SMR.1	FMT_SMR.1

Security functional requirement	Dependencies	Resolution
FMT_SMF_RMT.1	FMT_SMR.1	FMT_SMR.1
FMT_SMR.1	FIA_UID.1	FIA_UID.1

Table 11: TOE SFR dependency analysis

6.3 Security Assurance Requirements

The security assurance requirements for this evaluation are defined in [GPOSPP2] and will be taken for this evaluation as defined there.

The security assurance requirements (SARs) for the TOE are the components defined in the evaluation assurance package dummy.

The following table shows the SARs, and the operations performed on the components according to CC part 3: iteration (Iter.), refinement (Ref.), assignment (Ass.) and selection (Sel.).

Security assurance class	Security assurance requirement	Source	Operations			
			Iter.	Ref.	Ass.	Sel.
ASE Security Target evaluation	ASE_INT.1 ST introduction	CC Part 3	No	No	No	No
	ASE_CCL.1 Conformance claims	CC Part 3	No	No	No	No
	ASE_SPD.1 Security problem definition	CC Part 3	No	No	No	No
	ASE_OBJ.2 Security objectives	CC Part 3	No	No	No	No
	ASE_ECD.1 Extended components definition	CC Part 3	No	No	No	No
	ASE_REQ.2 Derived security requirements	CC Part 3	No	No	No	No
	ASE_TSS.1 TOE summary specification	CC Part 3	No	No	No	No
ADV Development	ADV_ARC.1 Security architecture description	CC Part 3	No	No	No	No
	ADV_FSP.1 Basic functional specification	CC Part 3	No	No	No	No
AGD Guidance documents	AGD_OPE.1 Operational user guidance	CC Part 3	No	No	No	No
	AGD_PRE.1 Preparative procedures	CC Part 3	No	No	No	No
ALC Life-cycle support	ALC_CMC.3 Authorisation controls	CC Part 3	No	No	No	No
	ALC_CMS.3 Implementation representation CM coverage	CC Part 3	No	No	No	No
	ALC_DEL.1 Delivery procedures	CC Part 3	No	No	No	No
	ALC_FLR.3 Systematic flaw remediation	CC Part 3	No	No	No	No
	ALC_LCD.1 Developer defined life-cycle model	CC Part 3	No	No	No	No

Security assurance class	Security assurance requirement	Source	Operations			
			Iter.	**Ref.**	**Ass.**	**Sel.**
ATE Tests	ATE_COV.2 Analysis of coverage	CC Part 3	No	No	No	No
	ATE_DPT.1 Testing: basic design	CC Part 3	No	No	No	No
	ATE_FUN.1 Functional testing	CC Part 3	No	No	No	No
	ATE_IND.2 Independent testing - sample	CC Part 3	No	No	No	No
AVA Vulnerability assessment	AVA_VAN.2 Vulnerability analysis	CC Part 3	No	No	No	No

Table 12: SARs

6.4 Security Assurance Requirements Rationale

The justification of the assurance requirements is given in [GPOSPP1].

7 TOE Summary Specification

7.1 General System Overview

The section about the general system overview describes the Linux operating system in general terms. It therefore also describes mechanisms which are not claimed by SFRs to support the general understanding of the system. With section 1.5.3, those unclaimed functions are properly identified.

The Target of Evaluation (TOE) is the Linux distribution in the version specified in the ST executing on the hardware specified in the ST.

Multiple TOE systems can be connected via a physically-protected Local Area Network (LAN).The environment is allows interconnecting a series of TOE systems. Each TOE system is executing the Linux operating system on hardware system identified in the ST. Each computer provides the same set of local services, such as file, memory, and process management. Each computer also provides network services, such as remote secure shells and file transfers, to users on other computers. A user logs in to a host computer and requests services from the local host and also from other computers within the LAN.

User programs issue network requests by sending Transmission Control Protocol (TCP) or User Datagram Protocol (UDP) messages to another computer. Some network protocols, such as Secure Shell (SSH), can start a shell process for the user on another computer, while others are handled by trusted server daemon processes.

The TOE system provides a user Identification and Authentication (I&A) mechanism by requiring each user to log in with proper password at the local workstation, and also at any remote computer where the user can enter commands into a shell program (for example, remote SSH sessions). Each computer enforces a set of the following policies:

- Discretionary Access Control (DAC) policy, based on UNIX®-style mode bits
- an optional Access Control List (ACL)
- Mandatory Access Control (MAC) policy using Security Enhanced Linux (SELinux) extensions

for the named objects under its control.

7.1.1 High-level product overview

The TOE system can be connected to other systems by a protected LAN. Linux provides a multi-user, multi-processing environment, where users interact with the operating system by issuing commands to a command interpreter. Users issue the commands by running system utilities, or by developing their own software to run in their own protected environments.

The following subsections present a structural overview of the hardware and software that make up an individual host computer. This single-computer architecture is one of the configurations permitted under this evaluation.

7.1.1.1 Host computer structure

This section describes the structure of Linux for an individual host computer. As shown in the following figure, the system consists of hardware, the Linux kernel, trusted non-kernel processes, TSF databases, and untrusted processes. In this figure, the TOE itself consists of Kernel Mode software, User Mode software, and hardware. The TOE Security Functions (TSF) are shaded in gray. Details such as interactions within the kernel, inter-process communications, and direct user access to the hardware are omitted.

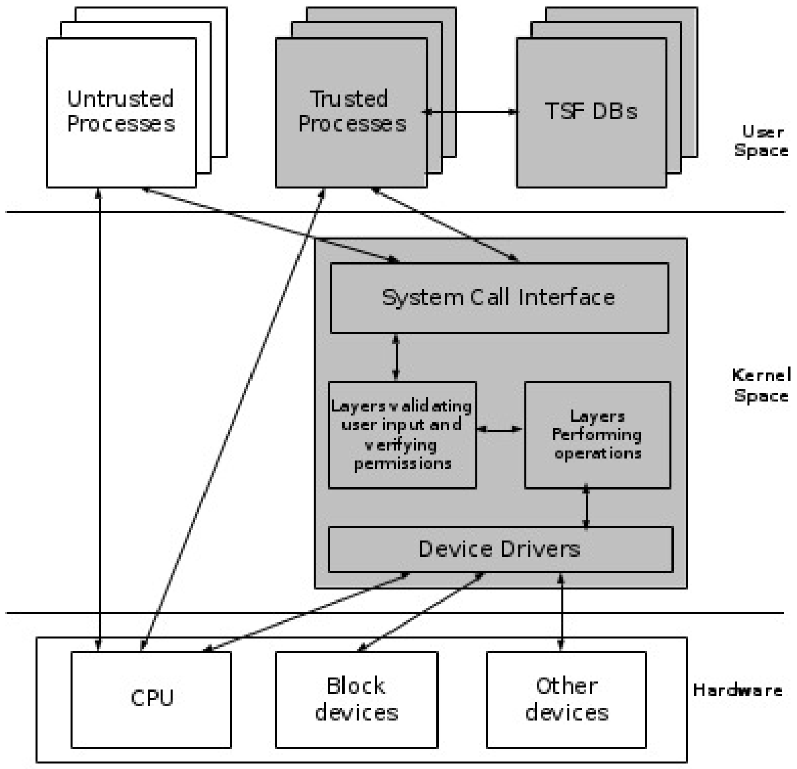


Figure 1: Overal Structure of TOE

The planar components, including CPUs, memory, buses, on-board adapters, and support circuitry; additional adapters, including LAN and video; and, other peripherals, including storage devices, monitors, keyboards, and front-panel hardware, constitute the hardware.

The Linux kernel includes the base kernel and separately-loadable kernel modules and device drivers. The kernel consists of the bootable kernel image and its loadable modules. The kernel implements the Linux system call interface, which provides system calls for file management, memory management, process management, networking, and other TSF (logical subsystems) functions addressed in the Functional Descriptions chapter of this document. An incoming request from user space passes the system call layer and it checked for its input and permission. After these checks, the requested work is executed which may need to access device drivers to obtain hardware access. The structure of the Linux kernel is described further in the Software Architecture chapter of this document.

Non-kernel TSF software includes programs that run with the administrative privilege, such as the sshd, and systemd daemons. The TSF also includes the configuration files that define authorized users, groups of users, services provided by the system, and other configuration data. Not included as TSF are shells used by administrators, and standard utilities invoked by administrators.

The Linux system, which includes hardware, kernel-mode software, non-kernel programs, and databases, provides a protected environment in which users and administrators run the programs, or sequences of CPU instructions. Programs execute as processes with the identity of the users that started them (except for some exceptions defined in this document), and with privileges as dictated by the system security policy. Programs are subject to the access control and accountability processes of the system.

7.1.1.2 System structure

The TOE permits one user at a time to log in to the computer's physical console. Several virtual consoles can be mapped to a single physical console. Different users can simultaneously login through different virtual consoles. The system can be connected to other computers via physically and logically protected LANs.

A standalone host configuration operates as a CC-evaluated system, which can be used by multiple users at a time. Users can operate by logging in at the virtual consoles or serial terminals of a system, or by setting-up background execution jobs. Users can request local services, such as file, memory, and process management, by making system calls to the kernel. Even though interconnection of different systems running the TOE is not included in the evaluation boundary, the networking software is loaded. This aids in a user’s request for network services (for example, SSH) from server processes on the same host.

Another configuration provides a useful network configuration, in which a user can log in to the console of any of the host computers, request local services at that computer, and also request network services from any of the other computers. For example, a user can use SSH to log into one host from another, or transfer files from one host to another. The configuration extends the single LAN architecture to show that Linux provides Internet Protocol (IP) routing from one LAN segment to another. For example, a user can log in at the console of a host in one network segment and establish an SSH connection to a host in another network segment. Packets on the connection travel across a LAN segment, and they are routed by a host in that segment to a host on another LAN segment. The packets are eventually routed by the host in the second LAN segment to a host on a third LAN segment, and from there are routed to the target host. The number of hops from the client to the server are irrelevant to the security provided by the system, and are transparent to the user.

The hosts that perform routing functions have statically-configured routing tables. When the hosts use other components for routing (for example, a commercial router or switches), then those components are assumed to perform the routing functions correctly, and do not alter the data part of the packets.

If other systems are to be connected to the network, with multiple TOE systems connected via a physically protected LAN, then they need to be configured and managed by the same authority using an appropriate security policy that does not conflict with the security policy of the TOE.

7.1.1.3 TOE services

Each host computer in the system is capable of providing the following types of services:

- Local services to the users who are currently logged in to the system using a local computer console, virtual consoles, or terminal devices connected through physically protected serial lines.
- Local services to the previous users via deferred jobs; an example is the cron daemon.
- Local services to users who have accessed the local host via the network using a protocol such as SSH, which starts a user shell on the local host.
- Network services to potentially multiple users on either the local host or on remote hosts.
- Virtualization environments are provided to allow untrusted software to execute in user state of the processor.

The following illustrates the difference between local services that take place on each local host computer, versus network services that involve client-server architecture and a network service layer protocol. For example, a user can log in to the local host computer and make file system requests or memory management requests for services via system calls to the kernel of the local host. All such local services take place solely on the local host computer and are mediated solely by trusted software on that host.

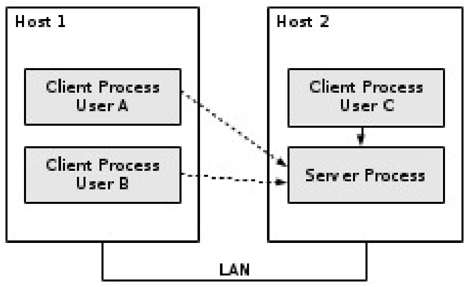


Figure 2: Local and network services provided by Linux

Network services, such as SSH or ftp, involve client-server architecture and a network service-layer protocol. The client-server model splits the software that provides a service into a client portion that makes the request, and a server portion that carries out the request, usually on a different computer. The service protocol is the interface between the client and server. For example, User A can log in at Host 1, and then use SSH to log in to Host 2. On Host 2, User A is logged in from a remote host.

On Host 1, when User A uses SSH to log in to Host 2, the SSH client on Host 1 makes protocol requests to an SSH server process on Host 2. The server process mediates the request on behalf of User A, carries out the requested service, if possible, and returns the results to the requesting client process.

Also, note that the network client and server can be on the same host system. For example, when User B uses SSH to log in to Host 2, the user's client process opens an SSH connection to the SSH server process on Host 2. Although this process takes place on the local host computer, it is distinguished from local services because it involves networking protocols.

7.1.1.4 Security policy

A user is an authorized individual with an account. Users can use the system in one of following ways:

- By interacting directly with the system through a session at a computer console (in which case the user can use the display provided as the physical console), or
- By interacting directly with system through a session at a serial terminal, or
- Through deferred execution of jobs using the cron mechanism, or
- By using services implemented with applications accessing these services either locally or remotely, or
- By using virtual machine environments accessing these environments either locally or remotely.

A user must log in at the local system in order to access the protected resources of the system. Once a user is authenticated, the user can access files or execute programs on the local computer, or make network requests to other computers in the system.

The only subjects in the system are processes. A process consists of an address space with an execution context. The process is confined to a computer; there is no mechanism for dispatching a process to run remotely (across TCP/IP) on another host. Every process has a process ID (PID) that is unique on its local host computer, but PIDs are not unique across systems. As an example, each host in the system has an init process with PID 1.

Objects are passive repositories of data. The TOE defines three types of objects:

- named objects which are resources, such as files and IPC objects, which can be manipulated by multiple users using a naming convention defined at the TSF interface;
- storage objects which is an object that supports both read and write access by multiple non-trusted subjects; and
- public objects which is an object that can be publicly read by non-trusted subjects and can be written only by trusted subjects.

Consistent with these definitions, all named objects are also categorized as storage objects, but not all storage objects are named objects.

Linux enforces a DAC policy for all named objects under its control, and an object reuse policy for all storage objects under its control. The DAC policy that is enforced varies among different object classes, in all cases it is based on user identity and on group membership associated with the user identity.

In addition to the DAC policy, Linux also enforces a MAC policy for all named objects under its control. DAC policy is enforced first, while MAC is enforced only if DAC permits the operation. The MAC policy is non-authoritative; that is, a DAC policy denial cannot be overridden by the MAC policy. The MAC policy that is enforced varies among different object classes, in all cases it is based on the domain of the user and the type of the object.

To allow for enforcement of the access control policies, all users must be identified, and their identities must be authenticated.

The TOE uses both hardware and software protection mechanisms. The hardware mechanisms used by Linux to provide a protected domain for its own execution include a multistate processor, memory segment protection, and memory page protection. The TOE software relies on these hardware mechanisms to implement TSF isolation, non-circumventability, and process address-space separation.

A user can log in at the console, at other directly attached terminals, or through a network connection. Authentication is based on a password entered by the user and authentication data stored in a protected file or via other types of credentials, such as cryptographic keys when using SSH. Users must log in to a host before they can access any named objects on that host. Some services, such as SSH, to obtain a shell prompt on another host, or ftp, to transfer files between hosts in the distributed system, require the user to re-enter authentication data to the remote host. Linux permits the user to change passwords (subject to TOE enforced password guidelines), change identity, submit batch jobs for deferred execution, and log out of the system.

The system architecture provides TSF self-protection and process isolation mechanisms.

7.1.1.5 Operation and administration

The network including the TOE can be composed of one, several, or many different host computers, each of which can be in various states of operation, such as being shut down, initializing, being in single-user mode, or online in a secure state. Thus, administration involves the configuration of multiple computers and the interactions of those computers, as well as the administration of users, groups, files, printers, and other resources for each host system.

The TOE provides commands which can be used to add, modify, and delete a user account. Also, the TOE provides commands to add, modify, and delete a group form the system, or to configure elements of the system security policy. These commands accept options to set up or modify various parameters for accounts, groups, and security policy. The commands modify the appropriate TSF databases and provide a safer way than manual editing to update authentication databases. Refer to the appropriate command man pages for detailed information about how to set up and maintain users and groups.

7.1.1.6 TSF interfaces

The TSF interfaces include local interfaces provided by each host computer, and the network client-server interfaces provided by pairs of host computers.

The local interfaces provided by an individual host computer include the following kernel-provided interfaces:

- System calls made by trusted and untrusted programs to the privileged kernel-mode software. As described separately in this document, system calls are exported by the base Linux kernel and by kernel modules.
- Although technically speaking the following kernel interfaces are a semantical extension of the open, ioctl or socket-related system calls, they should be considered independent interfaces during a security analysis:
    - Device files: device files allow direct access to hardware resources. The access is established using the read, write and/or mmap functions or using ioctls to implement more specific access mechanisms. In addition, a limited number of device files allow access to kernel-internal data structures using file system semantics as well as ioctls.
    - Virtual file systems: virtual file systems provide access to kernel-internal data structures using file system semantics. Access is limited to read, write and/or mmap functions.
    - Network sockets using the AF_NETLINK protocol: netlink sockets provide access to kernel-internal data structures using networking semantics. Access is limited to sending and/or receiving information.
    - Network sockets using the PF_KEY protocol: the PF_KEY sockets are used to interact with the network packet transformation logic in the kernel.
    - Network sockets using the AF_ALG protocol: The AF_ALG network protocol can be used to interact with the kernel crypto API.
- Any network protocol analyzer/parser implemented by the kernel. The following list enumerates the protocols available in the evaluated configuration:
    - Ethernet
    - ARP
    - TCP/IP protocol family

    - IPSec
    - Labeled communication using IPSec protocols

The following interfaces are implemented by trusted processes in user space:

- Networked interfaces provided by pairs of host computers:
    - SSHv2. For more detailed information about these interfaces, refer to: RFC 4252ff
    - IKEv1 and IKEv2: For more details on IKE refer to RFC 2409 as well as RFC 5996 – the available Diffie-Hellman groups are defined in RFC 2409, RFC 3526, RFC 5114.
    - TLS v1.1 and TLS v1.2: Details of the protocols are defined in RFC 4346 and RFC 5246.
- The argv and envp character arrays which are arguments to the execve system call. These two arrays can be evaluated by the executed application. Note that a number of environment variables that can be provided with envp are implemented by libraries that are commonly loaded by most, if not all, applications (like the C-library).
- Files that are part of the TSF database that define the configuration parameters used by the security functions.
- Inter-process communication interfaces exported by an application. This inter-process communication interface type includes the DBus support as DBus is a protocol that runs on top of kernel-provided inter-process communication mechanisms. The kernel inter-process communication mechanisms are agnostic of the DBus protocol specifics and are therefore secondary when assessing DBus.
- Hypervisor calls and instruction emulation and simulation implemented by the Linux kernel to support the KVM virtualization environment. This includes the simulation and emulation of hardware provided with the Linux kernel.

The following are interfaces that are not viewed as TSF interfaces:

- Interfaces between non-TSF processes and the underlying hardware. Typically, user processes do not interface directly with the hardware; exceptions are processor, USB, parallel port connections, serial port connections and graphics hardware. User processes interact with the processor by executing CPU instructions, reading and modifying CPU registers, and modifying the contents of physical memory assigned to the process. User processes interact with graphics hardware by modifying the contents of registers and memory on the graphics adapter. Accessing the remainder of the listed hardware from user space is done by using the proper signal lines to communicate with the respective device. Unprivileged processor instructions are externally visible interfaces. However, the unprivileged processor instructions do not implement any security functionality, and the processor restricts these instructions to the bounds defined by the processor. In addition, the rest of the listed hardware components are not part of the TOE as they are peripherals. Therefore, this interface is not considered as part of the TSF.
- Interfaces between different parts of the TSF that are invisible to normal users (for example, between subroutines within the kernel) are not considered to be TSF interfaces. This is because the interface is internal to the trusted part of the TOE and cannot be invoked outside of those parts. Those interfaces are therefore not part of the functional specification, but are explained in this HLD.
- The firmware, while part of the TOE, are not considered as providing TSF interfaces because they do not allow direct unprivileged operations to them.

- Processor exceptions reflected to the firmware, are not considered to be TSF interfaces. They are not relevant to security because they provide access to the firmware, which does not implement any security functionality.
- In case the IT environment provides a virtualization environment, such as KVM host systems, z/VM or PR/SM on s390x systems or POWER LPAR on IBM POWER systems, the architected hypervisor interfaces of the virtualization environment (like hypervisor calls) are not considered a TSF interface because it is not accessible by unprivileged processes in the problem state, and does not provide any security functionality. Note, the hypervisor functionality implemented by the Linux kernel and its interfaces is not referenced here.
- The SMM state of Intel-based processors is not security relevant as the software executing in this state does not implement any security functionality. Moreover, unprivileged code cannot access or modify the software executing in this processor state.

TSF interfaces include any interface that is possible between untrusted software and the TSF.

7.1.2 TSF identification

This section summarizes the approach to identification of the TSF.

The hardware and firmware are not considered providing TSF interfaces. The Linux operating system, on the other hand, does provide TSF interfaces.

The Linux operating system is distributed as a collection of packages. A package can include programs, configuration data, and documentation for the package. Analysis is performed at the file level, except where a particular package can be treated collectively. A file is included in the TSF for one or more of the following reasons:

- It contains code, such as the kernel, kernel module, and device drivers, that runs in a privileged hardware state of the processor.
- It enforces the security policy of the system.
- It allows SUID or SGID to a privileged user (for example, root) or group.
- It is given file system capabilities implying a privilege escalation during execution.
- It grants one or more abilities to override security-related rules enforced by the system to the calling user.
- It started as a daemon executing with root privileges or with a system UID / GID; an example is one started by systemd.
- It is software that must function correctly to support the system security mechanisms.
- It is required for system administration.
- It consists of TSF data or configuration files.
- It consists of libraries linked to TSF programs.
- It started as an application with different privileges than the caller; for example using mechanisms like udev, DBus, PolKit, modprobe, or even the kernelIt is very rare that the kernel performs callbacks to userspace. The most notable examples are init, modprobe, and v86d used by the uvesafb driver.
- It grants one or more MLS override capabilities to the calling user.

There is a distinction between non-TSF user-mode software that can be loaded and run on the system, and software that must be excluded from the system. The following methods are used to ensure that excluded software cannot be used to violate the security policies of the system:

- Addition of kernel modules is not permitted.

- The installation software may change the configuration (for example, mode bits) so that a program cannot violate the security policy.
- Addition of programs with SUID bit enabled and owning UID of either root or a system UID.
- Addition of programs with SGID bit enabled and owning GID of either root or a system GID.
- Addition of programs capability bits enabled.
- Addition of daemons executing with root or system UID / GID.
- Alterations of the rule set of the frameworks that allow spawning of processes with different privileges, such as udev or PolicyKit.
- Addition of programs with MLS override capabilities enabled.

7.2 Software architecture

This chapter summarizes the software structure and design of the Linux system and provides references to detailed design documentation.

The following subsections describe the TOE Security Functions (TSF) software and the TSF databases for the Linux system. The descriptions are organized according to the structure of the system and describe the Linux kernel that controls access to shared resources from trusted (administrator) and untrusted (user) processes. This chapter provides a detailed look at the architectural pieces, or subsystems, that make up the kernel and the non-kernel TSF. This chapter also summarizes the databases that are used by the TSF.

The Functional Description chapter that follows this chapter describes the functions performed by the Linux logical subsystems. These logical subsystems generally correspond to the architectural subsystems described in this chapter. The two topics were separated into different chapters in order to emphasize that the material in the Functional Descriptions chapter describes how the system performs certain key security-relevant functions. The material in this chapter provides the foundation information for the descriptions in the Functional Description chapter.

7.2.1 Hardware and software privilege

This section describes the terms hardware privilege and software privilege as they relate to the Linux operating system. These two types of privileges are critical for the Linux system to provide TSF self-protection. This section does not enumerate the privileged and unprivileged programs. Rather, the TSF Software Structure identifies the privileged software as part of the description of the structure of the system.

7.2.1.1 Hardware privilege

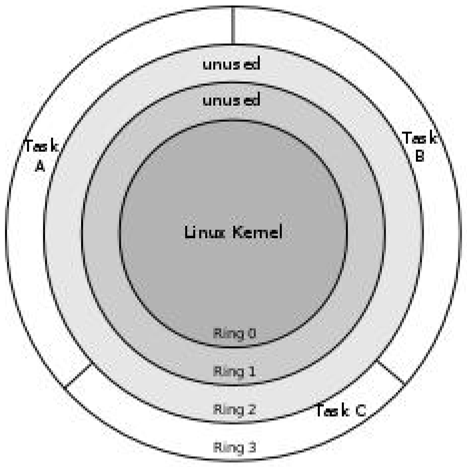
The underlying processors provides a notion of user mode execution and supervisor, or kernel, mode execution. The following briefly describes how these user- and kernel-execution modes are provided by the different architectures.

X86 Privilege level

This section describes the concept of privilege levels by using Intel-based / AMD-based processors as an example. The other processor architectures use a very similar concept albeit the protection mechanisms are called differently. For example, the PowerPC and System Z processors implement a supervisor and user state that are logically identical to ring 0 and 3 in Intel-based systems. The concept of privilege is implemented by assigning a value of 0 to 3 to key objects recognized by the processor. This value is called the privilege level. The following processor-recognized objects contain privilege levels:

- Descriptors contain a field called the descriptor privilege level (DPL).
- Selectors contain a field called the requestor's privilege level (RPL). The RPL is intended to represent the privilege level of the procedure that originates the selector.
- An internal processor register records the current privilege level (CPL). Normally the CPL is equal to the DPL of the segment the processor is currently executing. The CPL changes as control is transferred to segments with differing DPLs.

The following figure shows how these levels of privilege can be interpreted as layers of protection. The center is for the segments containing the most critical software, usually the kernel of the operating system. Outer layers are for the segments of less critical software.



The Linux kernel, as with most other UNIX-variant kernels, utilizes only two of these execution modes. The highest, with the processor privilege level of 0, corresponds to the kernel mode; the lowest, with the processor privilege of 3, corresponds to the user mode.

User and kernel modes, which are offered by all of the processors, implement hardware privilege as follows:

- When the processor is in kernel mode, the program has hardware privilege because it can access and modify any addressable resources, such as memory, page tables, I/O address space, and memory management registers. This is not possible in the user mode.
- When the processor is in kernel mode, the program has hardware privilege because it can execute certain privileged instructions that are not available in user mode.

Thus, any code that runs in kernel mode executes with hardware privileges. Software that runs with hardware privileges includes:

- The base Linux kernel. This constitutes a large portion of software that performs memory management file I/O and process management.
- Separately loaded kernel modules, such as many device driver modules. A kernel module is an object file whose code can be linked to, and unlinked from, the kernel at runtime. The kernel module code is executed in kernel like any other statically-linked kernel function.

All other software on the system normally runs in user mode, without hardware privileges, including user processes such as shells, networking client software, and editors. User-mode processes run with hardware privileges when they invoke a system call. The execution of the system call or processor traps (such as page faults) caused by user space applications switches the mode from user to kernel mode, and continues the operation at a designated address within the kernel where the code of the system call handler or trap handler is located.

The following paragraphs explain the hardware privileges for processor architectures other than the x86 architecture.

PowerPC Privilege level

This processor architecture provides three execution modes, identified by the PR bit (bit 49) and the HV bit (bit 3) of the Machine State Register (MSR) of the processor. Values of 0 for both PR and HV bits indicate a hypervisor execution mode. An HV bit value of 1, and a PR bit value of 0, indicate a supervisor, or kernel, execution mode. An HV bit value of 1 and a PR bit value of 1 indicate a user execution mode.

SystemZ Privilege level

The System z systems also provide two execution modes identified by the Problem State bit (bit 15) of the processor's Program Status Word (PSW). A value of 0 indicates a supervisor, or kernel, execution mode, and the value of 1 indicates a problem state, or user, execution mode.

Virtualization consideration

When the TOE is used as a host system for the KVM virtualization, a third privilege state is used in addition to the two mentioned above: the hypervisor mode. The hypervisor mode utilizes additional hardware support provided by the processor.

With the hypervisor mode, processor register are accessible that are not accessible via the other two states. Using these registers, another layer of memory address translation is implemented. In addition, I/O address space virtualization is utilized if available. The Linux kernel utilizes this mode when the KVM virtualization is activated. In this case, the entire Linux kernel operates in hypervisor mode.

Normal processes still operate in user mode. For user mode processes, the Linux kernel behaves the same way as if the kernel would operate in supervisor mode.

In addition, processes may also use the supervisor mode with the help provided by KVM to implement a guest operating system. From the Linux kernel perspective, a guest system is nothing more than a regular process where parts of that process are executed by enabling the virtualization functionality of the processor.

The following CPU mechanisms are used to implement the hypervisor state:

- Intel-based: VT-x
- AMD-based: SVM
- System Z: SIE instruction

The Linux kernel does not provide virtualization support for CPUs other than those listed.

7.2.1.2 Software privilege

Software privilege in Linux involves the ability to override the kernel's access control mechanisms. Linux implements the following access control models:

- Discretionary Access Control enforced on storage objects

- Capability-based security checks to restrict the execution of system calls or functional subsets of system calls to callers having the respective capability
- Linux Security Module (LSM) access checks – Mandatory Access Control policies are implemented using LSMs

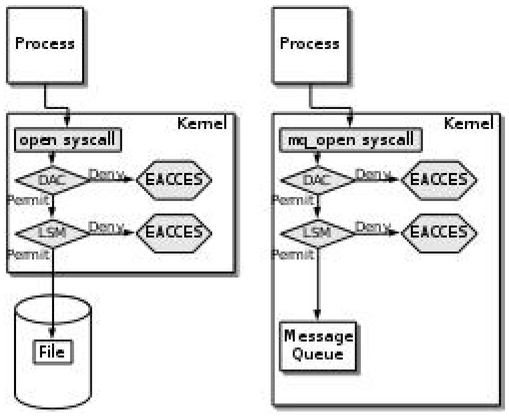
When accessing named objects, Discretionary Access Control (DAC) is applied first, and the LSM access checks is applied if and only if the DAC check grants access. The kernel implements software privileges for the DAC policy.

In addition, the SELinux LSM module may implement software privileges with an optionally loaded MLS policy.

Besides the access control software privileges, capabilities are another software privilege that can be granted to applications.

This section first describes the Linux access control model followed by the discussion of the capability model.

The following figure illustrates the access to a file system object as well as an IPC object and the access control enforcement. The illustration also shows that only the kernel is able to connect user space applications with either file system objects or IPC objects as user space is not allowed to directly access the storage locations.



As illustrated in the figure, access control is applied every time a subject attempts to access a storage object. Both subjects and objects carry security attributes that are relevant for the DAC policy and the access control policy enforced by the LSM. The kernel evaluates access based on these attributes and access control rules. If access is granted by DAC policy, the LSM hook is called to evaluate access based on the loaded LSM. Which LSM is loaded is subject to the configuration of the TOE.

DAC

The DAC model allows the owner of the object to decide who can access that object, and in what manner. Like any other access control model, DAC implementation can be explained by which subjects and objects are under the control of the model, security attributes used by the model, access control and attribute transition rules, and the override (software privilege) mechanism to bypass those rules.

Subjects and objects

Subjects in Linux are regular processes and kernel threads. They are both represented by the task_struct structure. Kernel threads run only in the kernel mode, and are not constrained by the DAC policy. All storage objects such as regular files, character and block files, directories, sockets, IPC objects, and kernel key rings are under the control of the DAC policy.

Attributes

Subject attributes used to enforce DAC policy are the process UID, GID, supplementary groups, and process capabilities. These attributes are stored in the task_structure of the process, and are affected by the system calls as described in Section 5.2. Object attributes used to enforce DAC policy are owner, group owner, permission bits, and POSIX.1e Access Control Lists (ACLs) for file system objects. These attributes are stored in-core and, for appropriate disk-based file systems, in the on-disk inode.

Access control rules

DAC access control rules specify how a certain process with appropriate DAC security attributes can access an object with a set of DAC security attributes. In addition, DAC access control rules also specify how subject and object security attributes transition to new values and under what conditions.

Software privilege

Software privilege for DAC policy is based on the capabilities of CAP_DAC_OVERRIDE, CAP_DAC_READ_SEARCH which can be assigned to a process. A process bearing these capabilities is granted software privilege with respect to DAC as it can bypass the access control policies of the system.

SELinux LSM MAC

With the Mandatory Access Control implemented with the SELinux LSM, it is the system security policy, unlike the owner in DAC, that controls who should be allowed access to what information.

The MAC policy provided with SELinux rests on two policy types:

- Type Enforcement (TE) which is used to implement role-based access control, and
- Multi-Level Security (MLS) with a Bell-LaPadula style hierarchical labeling policy.

This section describes the enforcement engine of the Linux MAC model by using the subjects and objects for which the MAC is performed, the attributes that are used for this check, access control rules, and override rules.

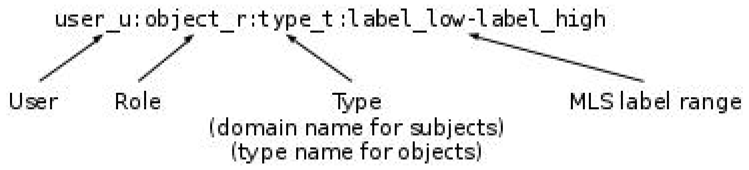
Subjects and objects

Subjects in Linux are regular processes and kernel threads. They are both represented by the task_struct structure. Kernel threads run only in the kernel mode and are not constrained by the MAC policy. All named objects such as regular files, character and block files, directories, sockets, and IPC objects are under the control of the MAC policy. In addition to named objects, the MAC policy can also control access to certain kernel data structures such as file descriptors, IPC messages, and file systems to allow granular expression of the system security policy.

Attributes

Subject and object security attributes, also referred to as the SELinux security context, have the same format. In user-readable and external form, the security context is an ASCII string. The kernel, for performance reasons, converts the string into a 32-bit integer at run-time; however, this document uses the ASCII version of the security context to describe the enforcement engine and the security policy.

The security context consists of four colon-separated components. They correspond to user, role, type, and MLS range of the subject or object as illustrated in the following figure.


user_u:object_r:type_t:label_low-label_high
User
Role
Type
(domain name for subjects)
(type name for objects)
MLS label range

- User: The user component is the SELinux user name.
- Role: The role component is the SELinux role corresponding to the subject or object. For subjects, the role is used to implement role-based access control by allowing the role to control access to domains. For objects, the role component is not used, and is all object_r.
- Type: The type field represents the subject domain or object type. Domains and types are equivalent classes for processes and resources, respectively. Access decisions in the kernel are made based on subject domain and object type.
- MLS label range: The MLS label range contains two complete MLS labels. They are arranged with the low label on the left and the high label, which dominates the low label, on the right. The two labels are separated by a dash and form a label range. For subjects, the low label of the range corresponds to the effective MLS label of the subject, while the high label corresponds to the clearance of the subject. The subject clearance maps to the clearance of the user on whose behalf the subject is acting.

For objects such as regular files, the security policy dictates that the low label and the high label are equal, thus making the object, effectively, single level. Objects such as directories, devices, and sockets may have a high label that is not equal to the low label. These objects are multilevel, and their MLS label range requires that the sensitivity level of any information that passes through them dominates the low label and is dominated by the high label.

Each MLS label consists of two components, a hierarchical classification level (or sensitivity level), and a non-hierarchical set of categories.

Between any two MLS labels, four possible relationships can exist, as follows whereas in the following listing L1 is the low label of the subject performing the access request and L2 is the low label of the object access is requested to:

- L1 is equal to L2 (levels are equal, category sets are equal)
- L1 dominates L2 (level of L1 >= level of L2, L2 category set equal or subset of L1 category set)
- L1 is dominated by L2 (level of L2 >= level of L1, L1 category set equal or subset of L2 category set)
- L1 is incomparable to L2 (L1 and L2 are not equal and neither dominates the other)

Attribute storage

SELinux attributes are stored:

- in the task_struct process structure for subjects,
- in in-core and on-disk inode data structure for file system objects.
- in the kern_ipc_perm data structure for system V IPC objects,
- in the sock data structure for sockets,
- in the xfrm_state structure for netlink sockets,
- and the key data structure for keys.

SELinux maps invalid contexts to the system_u:object_r:unlabeled_t:s0 context.

Access control rules

MAC access control implementation is based on Type Enforcement (TE). TE provides fine-grained access control over subjects and objects.

The TE and the MLS policy rules are defined with the SELinux policy. TE rules are defined as access control checks whereas the MLS policy is expressed as a constraint on top of the TE. Therefore, the total access control check happens in three logical steps, as follows:

1. The first is the DAC check using DAC attributes, followed by
2. the TE check using domains and types of the SELinux security context, followed by
3. the Bell-LaPadula MLS policy check using the MLS range of the SELinux security context.

Each check is non-authoritative. That is, permissions are only reduced at each stage; a DAC denial cannot be overridden by TE, and a TE denial cannot be overridden by MLS.

Software privilege

Software privilege for the MAC policy is implemented with the domain of the process and security policy rules associated with that domain.

For example, processes running in the init_t domain are allowed to read objects whose sensitivity label dominates the process sensitivity label. This override is achieved by domain attributes. The policy rule expresses the domain’s access to type with the override tied to an attribute of the domain. For example, the following policy rule can be used to allow a process to override the Bell-LaPadula restriction of “no-read-up” when it tries to read a regular file: mlsconstrain {file} {read} ((l1 dom l2) or (t1 == mlsfileread));

The above statement sets the security policy that label l1 (belonging to the subject) must dominate label l2 (belonging to the object) for the read operation to succeed, unless the process domain has the mlsfileread attribute.

System security policy for different subject and object types is described with the functional description of those subjects and objects.

In addition to the process attributes which can grant override rights, the following capability are known to the Linux kernel: CAP_MAC_OVERRIDE. If a process possesses this capability, the process can bypass the entire SELinux policy enforcement.

Capabilities

The Linux kernel has a framework for providing software privilege for many different functions provided by the Linux kernel by using capabilities. These capabilities, which are based on the POSIX.1e draft plus a number of additional capabilities defined and fully documented in include/linux/capability.h, allow breakup of the kernel software privilege associated with user ID zero into a set of discrete privileges based on the operation being attempted.

Capabilities integrate with user IDs as follows:

- If a SUID application with the owning ID of root is executed, all capabilities for that process are enabled.
- If the set*uid system call family is used to change the UID of the calling process to an ID not equal to 0, all capabilities are removed for that process.
- Processes spawned by root initially have all capabilities set.
- Processes spawned by non-root users initially have no capability set.

Software privilege

The kernel restricts the use of many system calls or functional areas called by system calls to processes which possess a specific capability. If the calling process does not possess the required capability, the execution of the affected code area is denied and the error of EPERM is returned to the caller.

For example, if a process is trying to create a device special file by invoking the mknod system call, the kernel checks to ensure that the process is capable of creating device special files by verifying that the process possesses the CAP_MKNOD capability. In the absence of special kernel modules that define and use capabilities, as is the case with the TOE, capability checks revert back to granting kernel software privilege based on the user ID of the process.

The entire list of all capabilities including their meanings is fully described in the Linux kernel source code file of include/linux/capability.h.

Programs with software privilege

Examples of programs running with software privilege are:

- Programs that are run by the system, such as the cron and init daemons.
- Programs that are run by trusted administrators to perform system administration.
- Programs that run with privileged identity by executing SUID / SGID / file system capabilities program files.
- Programs executed by trusted super daemons. The following super daemons with the capability of spawning applications with dissimilar privileged on behalf of calling users are present:
    - systemd: The system boot and management framework provided by systemd allows users to communicate with it via DBus channels. As systemd runs as root, it allows untrusted users to perform actions that otherwise would not be possible for such users.
    - PolKit: The user space authorization framework is implemented by the PolKit daemon. That daemon offers its services via DBus providing the capability to spawn applications as configured by its policy via a SUID helper application.

All software that runs with hardware privileges or software privileges are part of the TOE Security Functions (TSF).

In a properly administered system, unprivileged software is subject to the security policies of the system and does not have any means of bypassing the enforcement mechanisms. This unprivileged software need not be trusted in any way, and is thus referred to as untrusted software. Trusted processes that do not implement any security function need to be protected from unauthorized tampering using the security functions of Linux. They need to be trusted to not perform any function that violates the security policy of Linux.

7.2.2 TOE Security Functions software structure

This section describes the structure of the Linux software that constitutes the TOE Security Functions (TSF). The Linux system is a multi-user operating system, with the kernel running in a privileged hardware mode, and the user processes running in user mode. The TSF includes both the kernel software and certain trusted non-kernel processes.

The concept of breaking the TOE product into logical subsystems is described in the Common Criteria. These logical subsystems are the building blocks of the TOE, and are described in the Functional Descriptions chapter of this document. They include logical subsystems and trusted processes that implement security functions. A logical subsystem can implement or support one or more functional components. For example, the File and I/O subsystem is partly implemented by functions of the Virtual Memory Manager.

7.2.2.1 Kernel TSF software

The kernel is the core of the operating system. It interacts directly with the hardware, implements the sharing of resources, providing common services to programs, and prevents programs from directly accessing hardware-dependent functions. Services provided by the kernel include the following:

- Control of the execution of processes by allowing their creation, termination or suspension, and communication. These include:
  - Fair scheduling of processes for execution on the CPU.
  - Share of processes in the CPU in a time-shared manner.
  - CPU execution of a process.
  - Kernel suspension when its time quantum elapses.
  - Kernel schedule of another process to execute.
  - Later kernel rescheduling of the suspended process.
  - Management of the process security-related meta data, such as UIDs, GIDs, SELinux labels, capabilities.
- Allocation of the main memory for an executing process. These include:
  - Kernel allowance of processes to share portions of their address space under certain conditions, but protection of the private address space of a process from outside tampering.
  - If the system runs low on free memory, the kernel frees memory by writing a process temporarily to secondary memory, or a swap device.
  - Coordination with the machine hardware to set up a virtual-to-physical address that maps the compiler-generated addresses to their physical addresses.
- Life-cycle maintenance of virtual machines, which includes:

- Enforcement of the resource limits configured by the emulation application applicable to the virtual machine.
- Starting of the virtual machine code.
- Handling of exiting of virtual machines by either performing an instruction completion or deferring the instruction completion to the user-space emulation application.

- File system maintenance. These include:
  - Allocation of secondary memory for efficient storage and retrieval of user data.
  - Allocation of secondary storage for user files.
  - Reclamation of unused storage.
  - Structure of the file system in a well-understood manner.
  - Protection of user files from illegal access.
  - Allowance of processes' controlled access to peripheral devices such as terminals, tape drives, disk drives, and network devices.
  - Mediation of access between subjects and objects, allowing controlled access based on the DAC policy and any policy enforced by the loaded LSM.

The Linux kernel is a fully preemptible kernel. In non-preemptive kernels, kernel code runs until completion. That is, the scheduler is not capable of rescheduling a task while it is in the kernel. Moreover, the kernel code is scheduled cooperatively, not preemptively, and it runs until it finishes and returns to user-space, or explicitly blocks. In preemptive kernels, it is possible to preempt a task at any point, so long as the kernel is in a state in which it is safe to reschedule.

Logical components

The kernel consists of logical subsystems that provide different functionalities. Even though the kernel is a single executable program, the various services it provides can be broken into logical components. These components interact to provide specific functions.

The kernel consists of the following logical subsystems:

- File and I/O subsystem: This subsystem implements functions related to file system objects. Implemented functions include those that allow a process to create, maintain, interact, and delete file-system objects. These objects include regular files, directories, symbolic links, hard links, device-special files, named pipes, and sockets.
- Process subsystem: This subsystem implements functions related to process and thread management. Implemented functions include those that allow the creation, scheduling, execution, and deletion of process and thread subjects.
- Memory subsystem: This subsystem implements functions related to the management of memory resources of a system. Implemented functions include those that create and manage virtual memory, including management of page tables and paging algorithms.
- Networking subsystem: This subsystem implements UNIX and Internet domain sockets, as well as algorithms for scheduling network packets.
- IPC subsystem: This subsystem implements functions related to IPC mechanisms. Implemented functions include those that facilitate controlled sharing of information between processes, allowing them to share data and synchronize their execution, in order to interact with a common resource.

- Kernel framework subsystem: This subsystem implements the infrastructure for the kernel to sustain various kernel mechanisms. This subsystem includes the following functions among others:
    - Support for loadable modules: Implemented functions include those that load, initialize, and unload kernel modules.
    - Exception handling such as context switches, system call loading, etc.
    - Auditing: The audit subsystem implements functions related to recording of security-critical events on the system. Implemented functions include those that trap each system call to record security-critical events and those that implement the collection and recording of audit data.
- Linux Security Extensions: The Linux Security extensions implement various security-related aspects that are provided to the entire kernel, including the Linux Security Module framework. The LSM framework provides a security-agnostic framework for modules to implement different security policies, including SELinux. SELinux is an important logical subsystem. This subsystem implements mandatory access control functions to mediate access between all subjects and objects.
- Device driver subsystem: This subsystem implements support for various hardware and software devices through a common, device-independent interface.
- KVM subsystem: This subsystem implements the virtual machine life-cycle handling. It includes instruction completion for instructions requiring only small verifications. For any other instruction completion, KVM calls the QEMU user-space component.
- Crypto API: This subsystem provides a kernel-internal cryptographic library to all components of the kernel. It provides cryptographic primitives to callers.

Linux Containers are not considered to be a subsystem inside the Linux kernel. Linux Containers use different mechanisms provided by various subsystems. To form Linux Containers, namespaces, control groups and the seccomp filter mechanism are used. Namespaces are implemented by the subsystem for the respective type of namespace.

Execution components

The execution components of the kernel can be divided into three components: base kernel, kernel threads, and kernel modules, depending on their execution perspective.

Base kernel

The base kernel includes the code that is executed to provide a service, such as servicing a user’s system call invocation, or servicing an interrupt or exception event. A majority of the compiled kernel code falls under this category.

Kernel threads

In order to perform certain routine tasks such as flushing disk caches, or reclaiming memory by swapping out unused page frames, the kernel creates internal processes, or threads.

Threads are scheduled just like regular processes, but they do not have context in user mode. Kernel threads execute specific C kernel functions. Kernel threads reside in kernel space, and only run in the kernel mode.

Kernel modules and device drivers

Kernel modules are pieces of code that can be loaded and unloaded into and out of the kernel upon demand. They extend the functionality of the kernel without the need to reboot the system. Once loaded, the kernel module object code can access other kernel code and data in the same manner as statically-linked kernel object code.

A device driver is a special type of kernel module that allows the kernel to access the hardware connected to the system. These devices can be hard disks, monitors, or network interfaces. The driver interacts with the remaining part of the kernel through a specific interface, which allows the kernel to deal with all devices in a uniform way, independently of their underlying implementations.

7.2.2.2 Non-kernel TSF software

The non-kernel TSF software consists of trusted programs that are used to implement security functions. Note that shared libraries, including PAM modules in some cases, are used by trusted programs. However, there is no instance where a shared library by itself is considered to be a trusted entity. The trusted commands can be grouped as follows.

- System Initialization
- Identification and Authentication
- Network Applications
- Batch Processing
- System Management
- User Level Audit
- Cryptographic Support
- Virtual machine support
- User space authorization handling

7.2.2.3 TSF databases

Trusted databases are configuration files for trusted applications. None of these databases is modifiable by a user other than an administrative user. Access control is performed by the file system component of the Linux kernel. For more information about the format of these TSF databases, refer to their respective man pages.

7.2.3 Hardware

The hardware consists of the physical resources such as CPU, main memory, registers, caches, and devices that effectively make up the computer system.

7.2.4 Firmware

The firmware consists of the software residing in the hardware that is started when the system goes through a power-on reset. In addition to initializing the hardware and starting the operating system, on the partitioning-capable platforms the firmware provides logical partitioning support as well.

7.3 TOE Security Functionality

The following section explains how the security functions are implemented. The different TOE security functions cover the various SFR classes.

The primary security features of the TOE are:

- Audit
- Trusted Channel
- Network Information Flow Control
- Identification and Authentication
- Discretionary Access Control
- Security Management

7.3.1 Audit

The Lightweight Audit Framework (LAF) is designed to be an audit system for Linux compliant with the requirements from Common Criteria. LAF is able to intercept all system calls as well as retrieving audit log entries from privileged user space applications. The subsystem allows configuring the events to be actually audited from the set of all events that are possible to be audited. Those events are configured in a specific configuration file and then the kernel is notified to build its own internal structure for the events to be audited.

7.3.1.1 Audit functionality

The Linux kernel implements the core of the LAF functionality. It gathers all audit events, analyzes these events based on the audit rules and forwards the audit events that are requested to be audited to the audit daemon executing in user space.

Audit events are generated in various places of the kernel. In addition, a user space application can create audit records which needs to be fed to the kernel for further processing.

The audit functionality of the Linux kernel is configured by user space applications which communicate with the kernel using a specific netlink communication channel. This netlink channel is also to be used by applications that want to send an audit event to the kernel.

The kernel netlink interface is usable only by applications possessing the following capabilities:

- CAP_AUDIT_CONTROL: Performing management operations like adding or deleting audit rules, setting or getting auditing parameters;
- CAP_AUDIT_WRITE: Submitting audit records to the kernel which in turn forwards the audit records to the audit daemon.

Based on the audit rules, the kernel decides whether an audit event is discarded or to be sent to the user space audit daemon for storing it in the audit trail. The kernel sends the message to the audit daemon again using the above mentioned netlink communication channel. The audit daemon writes the audit records to the audit trail. An internal queuing mechanism is used for this purpose. When the queue does not have sufficient space to hold an audit record the TOE switches into single user mode, is halted or the audit daemon executes an administrator-specified notification action depending on the configuration of the audit daemon. This ensures that audit records do not get lost due to resource shortage and the administrator can backup and clear the audit trail to free disk space for new audit logs.

Access to audit data by normal users is prohibited by the discretionary access control function of the TOE, which is used to restrict the access to the audit trail and audit configuration files to the system administrator only.

The system administrator can define the events to be audited from the overall events that the Lightweight Audit Framework using simple filter expressions. This allows for a flexible definition of the events to be audited and the conditions under which events are audited. The system administrator is also able to define a set of user IDs for which auditing is active or alternatively a set of user IDs that are not audited.

The system administrator can select the audited events. Individual files can be configured to be audited by adding them to a watch list that is loaded into the kernel. In addition, audit rules can be specified to generate audit data based on a large number of different attributes, including:

- Subject or user identifiers
- Result of the operation (success/failure)
- Object identity
- Operation performed on an object
- System call number
- SELinux label components

The complete list of auditable operations can be obtained from the auditctl(8) man page.

The audit system can be configured to take actions if the audit trail is full or reaches a given theshold of disk space. The actions that can be configured include a halting of the system, preventing further auditable actions, notifications to an administrator or the execution of a configured command.

The TOE provides a management application that uses the aforementioned netlink interface. This application is used during boot time to load the audit rules from the configuration file /etc/audit/audit.rules. The audit rules can be modified at runtime of the system.

7.3.1.2 Audit trail

An audit record consists of one or more lines of text containing fields in a “keyword=value” tagged format. The following information is contained in all audit record lines:

- Type: indicates the source of the event, such as SYSCALL, PATH, USER_LOGIN, or LOGIN
- Timestamp: Date and time the audit record was generated
- Audit ID: unique numerical event identifier
- Login ID (“auid”), the user ID of the user authenticated by the system (regardless if the user has changed his real and / or effective user ID afterwards)
- Effective user and group ID: the effective user and group ID of the proces s at the time the audit event was generated
- Success or failure (where appropriate)
- Process ID of the subject that caused the event (PID)
- Hostname or terminal the subject used for performing the operation
- Information about the intended operation

This information is followed by event specific data. In some cases, such as SYSCALL event records involving file system objects, multiple text lines will be generated for a single event, these all have the same time stamp and audit ID to permit easy correlation.

The audit trail is stored in ASCII text. The TOE provides tools for managing ASCII files that can be used for post-processing of audit data. The application ausearch allows selective extraction of records from the audit trail using defined selection criteria. Using the ausearch, the administrator is able to select the information he wants to review. The tools allow the specification of a fine-grained search pattern where each information component can be searched for, including combinations of these patterns.

The audit trail is stored in files which are accessible by root only. If the audit trail fills up and reaches a warning threshold the administrator is notified about reaching the configured level. If the audit trail is full, the audit daemon rejects fetching new audit logs from the kernel to store them into a file. The kernel buffer holding audit messages fills up. When the kernel audit message buffer is full, the kernel suspends every subject that triggered an auditable event until the buffer is cleared again. This way, operations causing auditable events are prevented. In addition, the audit daemon can inform the administrator about the full audit trail, can switch to single user mode or halt the system, depending on the configuration.

This security function covers the SFRs of: FAU_GEN.1, FAU_GEN.2, FAU_SAR.1, FAU_SAR.2, FAU_STG.1.

7.3.1.3 Audit subsystem implementation

An auditing facility records information about actions that may affect the security of a computer system. In particular, an auditing facility records any action by any user that may represent a breach of system security. For each action, the auditing facility records enough information about those actions to verify the following:

- The user who performed the action
- The kernel object on which the action was performed
- The exact date and time it was performed
- The success or failure of the action
- The identity of the object involved

The TOE includes a comprehensive audit framework called Linux Audit Framework (LAF), which is composed of user-space and kernel-space components. The framework records security events in the form of an audit trail and provides tools for an administrative user. These tools enable the administrator to configure the subsystem and to search for particular audit records, providing the administrator with the ability to identify attempted and realized violations of the system’s security policy.

This section describes the design and operation of the audit subsystem at a high level.

Audit components

The following figure illustrates the various components that make up the audit framework and how they interact with each other. In general, there are user-space components and kernel-space components that use a netlink socket for communication. Whenever a security event of interest occurs, the kernel queues a record describing the event and its result to the netlink socket. If listening to the netlink, the audit daemon, auditd, reads the record and writes it to the audit log.

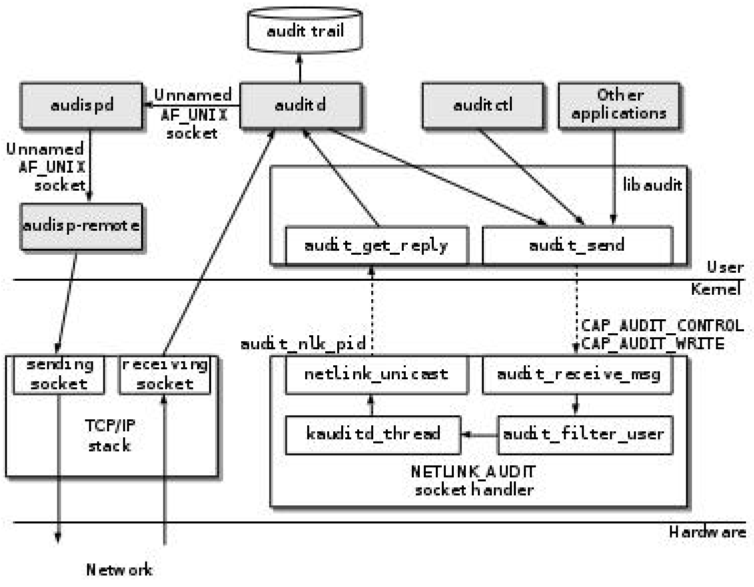


Figure 3: Audit framework

This section describes the various components of the audit subsystem, starting with the kernel components and then followed by the user-level components.

Kernel-userspace interface

On top of netlink, there exists the generic netlink family that provides simplified access for less demanding users. This introduces a control for ID management and name resolution, and possesses a new type of safety interface for netlink messages and attributes handling. This interface also features simplified message constructing, validation capabilities, and documentation.

This mechanism also receives user-space commands to control the operation of the audit framework and to set the audit filter rules and file system watch points.

When user space applications want to generate an audit entry, they also have to use the netlink interface to send the message to the kernel.

The kernel checks the effective capabilities of the sender process. If the sender does not possess the right capability (CAP_AUDIT_WRITE), the netlink message is discarded.

As outlined above, the kernel sends the completely formatted audit entry to the audit daemon for storage. The interface the kernel uses is also the same netlink mechanism. However, how does the kernel know to which process it has to send the message to? During startup time, the audit daemon opens the netlink socket and sends a specific control message with its PID to the kernel. That control message registers the PID with the kernel-internal audit mechanisms. From the time of the registering on, this PID is used as the receiver of kernel messages.

Task structure extensions for audit

The audit subsystem extends the task structure to potentially include an audit context. By default, on task creation, the audit context is built, unless specifically denied by the per-task filter rules. Then, during system calls, the audit context data is filled. The audit subsystem further extends the audit context to allow for more auxiliary audit information, which might be needed for specific audit events.

The following fields are part of the audit context:

- Login ID: Login ID is the user ID of the logged-in user. It remains unchanged through the setuid or seteuid system calls. Login ID is required to irrefutably associate a user with that user's actions, even across su(8) calls or use of SUID binaries. The Login ID is set by writing the ID to /proc/<PID>/loginuid, which is performed during login time with the pam_loginuid.so module. The loginuid file is only writable by root and is readable by everyone. The /proc file system triggers the kernel function audit_set_loginuid to set the login uid for the user in the audit context. From then on, this login uid is maintained throughout the session to trace back all operations done in the session to the login user.
- state: State represents the audit state that controls the creation of per-task audit context and filling of system call specifics in the audit context. It can take the following values:
    - AUDIT_DISABLED: Do not create per-task audit_context. No syscall-specific audit records will be generated for the task
    - AUDIT_SETUP_CONTEXT: Create the per-task audit_context, but don't necessarily fill it in a syscall entry time (i.e., filter instead).
    - AUDIT_BUILD_CONTEXT: Create the per-task audit_context, and always fill it in at syscall entry time. This makes a full syscall record available if some other part of the kernel decides it should be recorded.
    - AUDIT_RECORD_CONTEXT: Create the per-task audit_context, always fill it in at syscall entry time, and always write out the audit record at syscall exit time.
- in_syscall: States whether the process is running in a syscall versus in an interrupt.
- serial: A unique number that helps identify a particular audit record.Along with ctime, it can determine which pieces belong to the same audit record. The (timestamp, serial) tuple is unique for each syscall and it lives from syscall entry to syscall exit.
- ctime: Time at system call entry
- major: System call number
- argv array: The first 4 arguments of the system call.
- name_count: Number of names. The maximum defined is 20.
- audit_names: An array of audit_names structure which holds the data copied by getname.
- auditable: This field is set to 1 if the audit_context needs to be written on syscall exit.
- pwd: Current working directory from where the task has started.
- pwdmnt: Current working directory mount point. Pwdmnt and pwd are used to set the cwd field of FS_WATCH audit record type.
- aux: A pointer to auxiliary data structure to be used for event specific audit information.
- pid: Process ID.
- arch: The machine architecture.
- personality: The OS personality number.

- Other fields: The audit context also holds the various user and group real, effective, ser and file system id's: uid, euid, suid, fsuid, gid, egid, sgid, fsgid.

Syscall auditing

The audit framework is hooked into the system call glue code of the kernel which is part of the system call interrupt handling routine. Every time a system call is called by a process, the following two states are triggered by the system call glue code:

1. Upon entering the kernel realm but before the actual function implementing the invoked system call is called, a callback to the audit framework is made (audit_syscall_entry). This callback first verifies whether the system call is to be audited based on the audit rules. If it determines that the system call is to be audited, it retrieves the system call number, converts the arguments to an ASCII string to store them with the audit trail and obtains other information like the caller PID and its IDs.
2. After the function implementing the invoked system call completes, but before control is returned to user space, another audit hook (audit_syscall_exit) is called by the system call glue code. This hook code verifies whether there is data generated in step 1. If so, it receives the return code of the system call, stores it together with the initial information to complete the audit entry. This audit entry is now forwarded to the audit daemon via the netlink interface.

To bridge the gap between step one and two, the kernel audit framework uses the audit context registered with the task_struct. This audit context data structure is filled with the information obtained in step 1.

If an architecture implements the system call handling as a kernel-internal thread, the kernel must expect the possibility that the same process can issue another system call before the first is completed. In this case, the kernel uses the audit context pointer of the data structure and generates a double linked list with the pointer to the latest audit context structure as the head of the list. This list is walked during step 2 to find the right entry and merge the exit audit data with the right entry information.

Socket call and IPC audit record generation

Some system calls pass an argument to the kernel specifying which function the system call is requesting from the kernel. These system calls request multiple services from the kernel through a single entry point. For example, the first argument to the ipc call specifies whether the request is for semaphore operation, shared memory operation, and so forth. In the same manner, the socketcall system call is a common kernel entry point for the socket system calls. The socketcall and the ipc call are extended to audit the arguments and therefore audit the exact service being performed. Following is a typical flow:

1. The kernel encounters a socket or ipc call.
2. The kernel invokes an audit framework function to collect appropriate data to be used in the auxiliary audit context.
3. The call is processed.
4. On exit the audit record that includes the auxiliary audit information is placed on the netlink.

Filesystem auditing

File system auditing is implemented using of the inotify kernel file modification notification system. The audit_init kernel audit subsystem initialization routine registers a vector of inotify operations using the inotify_init function. The operations vector contains the audit_handle_ievent audit subsystem inotify event notification function and the audit_free_parent audit subsystem inotify destroy function. The audit subsystem inotify handle is returned by a successful audit_init call. When audit inotify events occur, audit_handle_ievent updates audit context inode data to reflect changes in watched file status.

When the audit subsystem receives an instruction from auditctl to set a watch on a file system object, the audit_recieve_skb function receives the netlink packet in the kernel. It in turn calls audit_receive_message, which dispatches the appropriate function based upon the operation requested. For audit rule updates, it calls audit_receive_filter. The audit_receive_filter routine calls audit_data_to_entry, which converts the audit data to a watch and calls audit_to_watch to initialize the audit watch data structure, and then calls audit_add_rule. The audit add_rule_function adds the inotify watch for the watch rule by calling audit_add_watch, which scans the list of active audit inotify watch parents and adds the parent if it does not already exist by calling audit_init_parent. The audit_init_parent function calls inotify_init_watch and inotify_add_watch to initialize the inotify watch and register it with the inotify subsystem. It finally adds the watch to the parent by calling the audit_add_to_parent function, which associates the watch rule with the watch parent.

When a filesystem object the audit subsystem is watching changes, the inotify subsystem calls the audit_handle_ievent function. audit_handle_ievent in turn updates the audit subsystem's watch data for the watched entity.

Permission changes, as well as access and modification of the object security attributes chown, chmod, setxattr, and removexattr, are audited by audit_inode hooks inserted into the system calls. The hooks directly update the inode information in the audit context.

When a watched object is accessed by a system call, the audit subsystem's information about the inode and its watches is updated. A typical sequence of file system operations that generates audit records for a watched object follows these steps:

1. A system call is entered.
2. The system call modifies a watched file's inode information, triggering an inotify event that calls the audit_handle_ievent function with the inotify watch event information, which updates the audit context's inode information. In certain cases, a hooked system call updates the audit context's inode information.
3. At syscall exit, audit_log_exit detects the updated inode information in the audit context and emits PATH and SYSCALL records for the watch event via the audit netlink interface.

Auditing of other kernel actions

In addition to the auditing of system calls and file system objects, the audit mechanism inside the kernel provides service functions for any other functional area inside the kernel. These service functions can be used to generate an audit entry with arbitrary contents. That audit entry is forwarded, like any other audit entry, to the auditd daemon for storage.

Kernel audit initialization

At kernel startup four lists are created to hold the filter rules. One list is checked at task creation, another is checked at syscall entry time, the third is checked at syscall exit time, and the fourth is used to filter user messages. These lists hold the filter rules set by user-space components. Multiple variables are used to control the operation of audit.

During boot time, the audit enabled flag is set according to audit_default or to the boot parameter audit. No syscall or file system auditing takes place without audit_enabled being set to true.

The file system auditing is initialized by creating the watch lists and the hash table for the file system auditing.

Audit record format

Each audit record consists of the type of record, a time stamp, login ID, and process ID, along with variable audit data depending on the audit record type. In other words, the record depends on the audit event. Since audit records are written to user-space as soon as they are generated, a complete audit record might be written in several pieces. A time stamp and a serial number pair identify the various pieces of the audit records. The timestamp of the record and the serial number are used by the user-space daemon to determine which pieces belong to the same audit record. The tuple is unique for each syscall and lasts from syscall entry to syscall exit. The tuple is composed of the timestamp and the serial number.

Each audit record for system calls contains the system call return code, which indicates if the call was successful or not. The following table lists security-relevant events for which an audit record is generated on the TOE.

Event description	**LAF audit events**
Startup and shutdown of audit functions	DAEMON_START, DAEMON_END, generated by auditd
Modification of audit configuration files	DAEMON_CONFIG, DAEMON_RECONFIG generated by auditd. Syscalls open, link, unlink, rename, truncate, (write access to configuration files)
Successful and unsuccessful file read/write	Syscall open
Audit storage space exceeds a threshold	Space_left_action, admin_space_left_action configuration parameters for auditd.
Audit storage space failure	Disk_full_action, disk_error_action configuration parameters for auditd.
Operation on file system objects and IPC objects	system calls accessing the objects
Rejection or acceptance by the TSF of any tested secret.	Audit record type: USER_AUTH from PAM Framework and audit record type: USER_CHAUTHTOK
Use of identification and authentication mechanism	Audit record type: USER_AUTH, USER_CHAUTHTOK from PAM framework.
Success and failure of binding user security attributes to a subject (e.g. success and failure to create a subject)	Audit record type: LOGIN from pam_login.so module. Syscalls: fork and clone.
All modification of subject security values	Syscalls chmod, chown, setxattr, msgctl, semctl, shmctl, removexattr, truncate

Event description	**LAF audit events**
Modifications of the default setting of permissive of restrictive rules	Syscalls umask, open
Modification of TSF data	System calls to access file system objects; audit record type: USER_CHAUTHTOK
Modifications to the group of users that are part of a role	Audit messages from trusted programs in the shadow suite, audit record type: USER_CHAUTHTOK.
Changes to system time	Syscall settimeofday, adjtimex; execution of hwclock and access to /dev/rtc

Auditing Support for IPTables

IPTables supports the creation of audit logs based on IPTables rules by providing an AUDIT target. That target can be added to generate an audit record for accepted, denied or rejected traffic. The AUDIT target creates an audit entry for the logged IP packet with the following information, if applicable:

- Netfilter hook
- packet length
- incoming/outgoing interface
- MAC source and destination address and protocol for Ethernet packets
- Source and destination address and protocol for IPv4/IPv6
- Source and destination port for TCP/UDP/UDPLITE
- ICMP type/code
- IPTables buffer marker

Auditing Support for OpenSSH

The OpenSSH server generates audit records for the following operations:

- The audit records contain an identifier that the sshd process generated the audit records and therefore implicitly identifying the used communication protocol.
- Origin of the communication channel by logging the remote IP and remote port are logged.
- Indication of a success establishment of a connection is logged. Note, the absence of that log entry indicates a failure of establishing a communication channel.
- Indication when a connection is terminated is logged.
- Authentication of a user (success and failure) including the user name is logged.
- Authentication type is logged (such as password-based or key-based authentication).
- The OpenSSH server logs cryptographic information of key exchange mechanism and the used user or host based authentication mechanisms. In addition, the server logs when a new ephemeral session key is established.
- If the server executes a command, this command will be logged.

Time Stamp Maintenance

The Linux kernel maintains various time stamps which has the following properties:

- Time with a resolution in seconds is available to user space. The time is obtained from the firmware or hardware at boot time.
- Time with a nanosecond resolution since the system started.
- Time with a microsecond resolution since Epoch (01.01.1970).

The auditing mechanism uses the available time information to add a time stamp to each audit record.

The configuration files including the auditd.conf and the audit.rules files for the audit framework covering all management aspects are writable by the root user only.

This security function covers the SFRs of: FAU_GEN.1, FAU_SEL.1, FAU_STG.3, FAU_STG.4, FPT_STM.1.

7.3.2 Trusted Channel

The TOE offers different cryptographic services to protect user data. The following subsections cover the different types of cryptographic services analyzed as part of the evaluation. Additional cryptographic mechanisms are active in the TOE which, however, are not subject to the assessments of this evaluation.

7.3.2.1 Cryptographic network services

The TOE provides cryptographically secured network communication channels to allow remote users to interact with the TOE. Using one of the following cryptographically secured network channels, a user can request the following services:

- OpenSSH: The OpenSSH application provides access to the command line interface of the TOE. Users may employ OpenSSH for interactive sessions as well as for non-interactive sessions. The console provided via OpenSSH provides the same environment as a local console. OpenSSH implements the SSHv2 protocol.

In addition to the cryptographically secured communication channels, the TOE also provides cryptographic algorithms for general use.

The cryptographic primitives for implementing the above mentioned cryptographic communication protocols are provided by OpenSSL.

SSHv2 Protocol

The TOE provides the Secure Shell Protocol Version 2 (SSH v2.0) to allow users from a remote host to establish a secure connection and perform a logon to the TOE.

The following table documents implementation details concerning the OpenSSH implementation's compliance to the relevant standards. It addresses areas where the standards permit different implementation choices such as optional features.

Reference	Description	Implementation Details
[RFC4253] chapter 5	Compatibility with old SSH versions	The OpenSSH implementation is capable of interoperating with clients and servers using the old 1.x protocol. That functionality is explicitly disabled in the evaluated configuration, it permits protocol version 2.0 exclusively.
[RFC4253] section 6.2	Compression	OpenSSH supports the OPTIONAL "zlib" compression method.

Reference	Description	Implementation Details
[RFC4253]☝ section 6.3	Encryption	The ciphers supported in the evaluated configuration are listed in FTP_ITC.1 for the SSH protocol.
[RFC4252]☝ chapter 7	Public Key Authentication Method: "publickey"	This REQUIRED authentication method is supported by OpenSSH but can be disabled by the administrator of the OpenSSH daemon.
[RFC4252]☝ chapter 8	Password Authentication Method: "password"	This SHOULD authentication method is supported by OpenSSH but can be disabled by the administrator of the OpenSSH daemon.
[RFC4252]☝ chapter 8	Password change request and setting new password	The OpenSSH implementation supports the optional password change mechanism in the evaluated configuration.
[RFC4252]☝ chapter 9	Host-Based Authentication: "hostbased"	This OPTIONAL authentication method is disabled in the evaluated configuration.

Table 13: SSH implementation notes

The TOE supports the generation of RSA, DSA and ECDSA key pairs. These key pairs are used by OpenSSH for the host keys as well as for the per-user keys. When a user registers his public key with the user he wants to access on the server side, a key-based authentication can be performed instead of a password-based authentication. The key generation mechanism uses the Linux kernel random number generator. The evaluated configuration permits the import of externally-generated key pairs.

This security function covers the SFRs of: FIA_PK_EXT.1, FTP_ITC.1.

The TOE supports the following security functions of the SSH v2.0 protocol:

- Establishing a secure communication channel using the following cryptographic functions provided by the SSH v2.0 protocol:
  - Encryption as defined in section 4.3 of [RFC4253]☝ - the keys are generated using the random number generator of the underlying cryptographic library;
  - Diffie-Hellman key exchange as defined in section 6.1 of [RFC4253]☝;
  - The keyed hash function for integrity protection as defined in section 4.4 of [RFC4253]☝.

  Note: The protocol supports more cryptographic algorithms than the ones listed above. Those other algorithms are not covered by this evaluation and should be disabled or not used when running the evaluated configuration.
- Performing user authentication using the standard password-based authentication method the TOE provides for users (password authentication method as defined in chapter 5 of [RFC4252]☝).
- Performing user authentication using a RSA, DSA or ECDSA key-based authentication method (public key authentication method as defined in chapter 5 of [RFC4252]☝).
- Checking the integrity of the messages exchanged and close down the connection in case an integrity error is detected.

The OpenSSH applications of sshd, ssh and ssh-keygen use the OpenSSL random number generator seeded by /dev/random or /dev/urandom to generate cryptographic keys. OpenSSL provides different DRNGs depending whether the FIPS 140-2 mode is enabled in the system.

OpenSSH Implementation Details

Secure Shell (SSH) is a network protocol that provides a replacement for insecure remote login and command execution facilities such as telnet, rlogin, and Remote Shell (RSH). SSH encrypts traffic, preventing traffic sniffing and password theft.

On a local system, the user starts the SSH client to open a connection to a remote server running the sshd daemon. If the user is authenticated successfully, an interactive session is initiated, allowing the user to run commands on the remote system. SSH is not a shell in the sense of a command interpreter, but it permits the use of a shell on the remote system.

In addition to interactive logins, the user can tunnel TCP network connections through the existing channel, allowing the use of X11 and other network-based applications, and copy files through the use of the scp and sftp tools. OpenSSH is configured to use the PAM framework for authentication, authorization, account maintenance, and session maintenance. Password expiration and locking are handled through the appropriate PAM functions.

Communication between the SSH client and SSH server uses the SSH protocol, version 2.0. The SSH protocol requires that each host have a host-specific key. When the SSH client initiates a connection, the keys are exchanged using the Diffie-Hellman protocol. A session key is generated, and all traffic is encrypted using this session key and the agreed-upon algorithm.

Default encryption algorithms supported by SSH are 3DES (triple DES) and blowfish. The default can be overridden by providing the list in the server configuration file with the “ciphers” keyword.

The default message authentication code algorithms supported by SSH are SHA-1 and MD5. The default can be overridden by providing the list in the server configuration file with the keyword MACs.

Encryption is provided by the OpenSSL package, which is a separate software package. The following briefly describes the default SSH setup with respect to encryption, integrity check, certificate format, and key exchange protocol.

- Encryption: A number of ciphers and block chaining modes are available with OpenSSH. A subset is allowed in the evaluated configuration.
- Integrity check: Data integrity is protected by including a message authentication code (MAC) with each packet that is computed from a shared secret, packet sequence number, and the contents of the packet. The message authentication algorithm and key are negotiated during key exchange. Initially, no MAC will be in effect, and its length must be zero. After key exchange, the selected MAC will be computed before encryption from the concatenation of packet data

  mac = MAC (key, sequence_number || unencrypted_packet) where unencrypted_packet is the entire packet without MAC (the length fields, payload and padding), and sequence_number is an implicit packet sequence number represented as uint32. The sequence number is initialized to zero for the first packet, and is incremented after every packet, regardless of whether encryption or MAC is in use. It is never reset, even if keys or algorithms are renegotiated later. It wraps around to zero after every 2^32 packets. The packet sequence number itself is not included in the packet sent over the wire.

The MAC algorithms for each direction must run independently, and implementations must allow choosing the algorithm independently for both directions. The MAC bytes resulting from the MAC algorithm must be transmitted without encryption as the last part of the packet. The number of MAC bytes depends on the algorithm chosen. The default MAC algorithm defined is the hmac-sha1 (with digest length = key length = 20 bytes).

- Certificate format: The default certificate format used is ssh-dss signed with Simple DSS. Signing and verifying using this key format is done according to the Digital Signature Standard [FIPS-186] using the SHA-1 hash. In addition to DSS, RSA, ECDSA are available.
- Key exchange protocol: The default key exchange protocol is diffie-hellman-group1-sha1. The diffie-hellman-group1- sha1 method specifies Diffie-Hellman key agreement with SHA-1 as HASH. The domain parameters of size 1024 bits are specified in [RFC2409] In addition, the following key agreement protocols are available:
  - diffie-hellman-group14-sha1: Diffie-Hellman key agreement with SHA-1 and domain parameters of size 2048 bits defined in RFC3526
  - diffie-hellman-group-exchange-sha1: Diffie-Hellman key agreement with SHA-1 and domain parameters generated as defined in RFC4419 - OpenSSH provides a set of pre-computed Diffie-Hellman domain parameters in /etc/ssh/moduli. During the SSH protocol handshake, the client and server negotiate the domain parameter set where both must agree on a set that is located in /etc/ssh/moduli on both sides.
  - diffie-hellman-group-exchange-sha256: This option is identical to diffie-hellman-group-exchange-sha1 except that it requires SHA-256 to be used.
  - ecdh-sha2-nistp256: Elliptic Curve Diffie-Hellman key agreement with SHA-256 using the NIST curve P-256 as defined in RFC5656
  - ecdh-sha2-nistp384: Elliptic Curve Diffie-Hellman key agreement with SHA-384 using the NIST curve P-384 as defined in RFC5656
  - ecdh-sha2-nistp521: Elliptic Curve Diffie-Hellman key agreement with SHA-512 using the NIST curve P-521 as defined in RFC5656
  - curve25519-sha256@libssh.org: Elliptic Curve Diffie-Hellman key agreement with SHA-256 using the Montgomery curve 25519 as defined in http://cr.yp.to/ecdh/curve25519-20060209.pdf

The following subsections briefly describe the implementation of SSH client and SSH server. For detailed information about the SSH Transport Layer Protocol, SSH Authentication Protocol, SSH Connection Protocol, and SSH Protocol Architecture, refer to the corresponding protocol specifications in RFC 4250ff.

SSH client

The SSH client first parses arguments and reads the configuration (readconf.c), then calls ssh_connect (in sshconnect*.c) to open a connection to the server, and performs authentication (ssh_login in sshconnect.c). Terminal echo is turned off while users type their passwords, which prevents the password from being displayed on the terminal as it is being typed. The SSH client then makes requests such as allocating a pseudo-tty, forwarding X11 connections, forwarding TCP-IP connections and so on, and might call code in ttymodes.c to encode current tty modes. Finally, the SSH client calls client_loop in clientloop.c.

The client is typically installed with suid as root. The client temporarily gives up this right while reading the configuration data. The root privileges are used to make the connection from a privileged socket, which is required for host-based authentication and to read the host key for host-based authentication using protocol version 1. Any extra privileges are dropped before calling ssh_login. Because .rhosts support is not included in the TSF, the SSH client is not suid root on the system.

SSH server daemon

The sshd daemon starts by processing arguments and reading the /etc/ssh/sshd_config configuration file. The configuration file contains keyword-argument pairs, one per line. Refer to the sshd_config man page for available configuration options. The daemon then reads the host key, starts listening for connections, and generates the server key.

When the server receives a connection, it forks a process and re-executes the sshd binary, disables the regeneration alarm, and starts communicating with the client. The server and client first perform identification string exchange, and then negotiate encryption and perform authentication. If authentication is successful, the forked process sets the effective user ID to that of the authenticated user, performs preparatory operations, and enters the normal session mode by calling server_loop in serverloop.c.

When the server accepts a new connection, it prints the contents of the file pointed to by the configuration variable “Banner” before any authentication takes place.

The sshd daemon in the TOE supports extended user/role/range@hostname login syntax for MLS role and level selection. Because newrole is restricted to administrative users, the extended role and level selection syntax is the method by which ordinary ordinary users select role and level.

Password-based authentication

The password based authentication utilizes the PAM library if the configuration option UsePAM is set in sshd_config. The SSH daemon receives the user name and password after setting up the SSH tunnel and feeds it into the PAM library. The following sequence is used by the SSH daemon to access the PAM library:

1. Initializing the interaction with the PAM library using the pam_start. The PAM configuration name is set to the file name of the SSH daemon which is “sshd”.
2. Establishing a thread that is used for the authentication conversation. That thread uses pam_authenticate to authenticate the user. If the PAM library requires a change of the authentication token, pam_chauthtok is called.
3. If the authentication returns PAM_SUCCESS, pam_open_session is used to set up the user session.

Key-based authentication

If the key-based authentication is enabled, the SSH daemon allows the use of RSA or DSA keys as authentication token.

The following steps are performed by the SSH daemon:

1. Verify that the user name is defined on the local system. If not, the authentication attempt is terminated.
2. The key-based authentication is performed as defined by RFC 4252. The public key for the key-based authentication must reside in the home directory of the target user in the file .ssh/authorized_keys. As this file may contain multiple key, each key is tried whether it is

appropriate as a public key for the authentication attempt (i.e. whether the public key can decrypt the data sent by the client encrypted with the client's private key). The first key that is found to match the private key indicates a successful authentication.

3. If the authentication was successful, pam_open_session is used to set up the user session. The session part of the PAM configuration file for the SSH daemon is applied.

This security function covers the SFRs of: FTP_ITC.1, FMT_SMF_RMT.1.

7.3.3 Network Information Flow Control

The Linux kernel's network stack implementation follows the layering structure of the network protocols. It implements the code for handling the link layer as well as the network layer. For those layers, independent filter mechanism are provided:

- Network layer: netfilter/iptables implements the filtering mechanism for non-bridge interfaces

Packet filter rules can only be injected into the Linux kernel for enforcement by processes possessing the CAP_NET_ADMIN capability.

7.3.3.1 Network layer filtering

Netfilter

Netfilter is a framework for packet mangling, implemented in the Linux kernel network stack handling the network layer. The netfilter framework comprises of the following parts:

- The IP stack defines five hooks which are well-defined points in a network packet's traversal of the IP protocol stack. Each of the hooks, the network stack will call the netfilter framework allowing it to operate on the entire packet. Note: the netfilter framework provides such hooks in a number of network protocol implementations, but the TOE only supports IP as outlined above. Therefore, the ST specification only covers the IP protocol.
- The netfilter framework provides register functions for other kernel parts to listen to the different hooks. When a packet traverses one of the hooks and passed to the netfilter framework, it invokes every registered kernel part. These kernel parts then can examine the packet and possible alter it. As part of the examination, these kernel parts can instruct the netfilter framework to discard the packet, to allow it to pass, or to queue it to user space.
- When a packet is marked to be queued to user space, the netfilter framework handles the asynchronous communication with user space.

The netfilter framework implements the five hooks at the following points in the packet traversal chain:

- When the packet enters the network layer of the TOE and after applying some sanity checks, but before the routing table is consulted, the NF_IP_PRE_ROUTING hook is triggered.
- After passing the routing table decision and the routing code marks the packet to be targeted for another host, the NF_IP_FORWARD hook is triggered.
- After passing the routing table decision and the routing code marks the packet to be targeted for the local system, the NF_IP_LOCAL_IN hook is triggered.
- When the packet traversed all of the network stack and is about to be placed on the wire again, the NF_IP_POST_ROUTING hook is triggered.

- When a packet is generated locally, the NF_IP_LOCAL_OUT hook is triggered before the routing table is consulted.

IPTables

All communication on the network layer can be controlled by the IPTables framework.

The TOE implements a packet filter as part of the network stack provided with the Linux kernel. The combination of IPTables and netfilter implements the packet filter which provides stateful and stateless packet filtering for network communication by inspecting the IP header, the TCP header, UDP header and/or ICMP header of every network packet that passes the network stack.

The packet selection system called IP Tables uses the netfilter framework to implement the actual packet filtering logic on the network layer for the TCP/IP protocol family.

Note: IPTables is able to perform Network Address Translation (NAT) as well as Port Address Translation (PAT) for simple as well as more complex protocols. This mechanism is out of scope for the evaluation. Furthermore, packet mangling support is provided with IPTables which is also out of scope for the evaluation.

IPTables registers all hooks provided by the netfilter framework. The NAT/PAT mechanism uses the pre-routing and post-routing hooks whereas the packet filtering capability is enforced on the local-in, local-out and forwaring hooks.

IPTables consists of the following two components:

- In-kernel packet filter enforcement: The kernel-side of IPTables use the netfilter framework as indicated above. Three lists of packet filter rules are enforced by the kernel mechanism: one for each netfilter framework hook that applies to packet filtering. When a packet is analyzed by the IPTables kernel modules, they first select the applicable list based on the hook where the netfilter framework triggered IPTables. Each list contains zero or more rules which are iterated sequentially. A rule consists of a matching part (also called the "match extension") and an action part (also called the "target extension"). When a rule is applied to a packet, the kernel modules first applies the matching part of the rule. If the packet matches, the action part is enforced. If the action part contains a decision of the fate of the packet (to accept it, to drop it, or to drop it and sending a notification to the sender), the rule list validation stops for this packet. If the action part contains a modification instruction or log instruction for the packet, the rule list validation continues after performing this operation. When the rule list is iterated through and a packet could not be matched by a rule with a decision action (accept, drop), the default decision action applicable to the list is enforced. This default action is either to accept the packet, to drop the packet, or to drop the packet and send a notification to the sender.
- User space configuration application: The user space application [IPTABLES] supported by [IPTABLES-EXT] allows the configuration of the IPTables kernel components. The application allows the specification of one rule per invocation where a rule contains the above mentioned matching part and action part. The tool also allows modification or deletion of existing rules as well as configuration of the default action. When using the tool, each invocation must specify the netfilter framework hook to which the rule applies to. See the man page of iptables(1) for more details.

This security function covers the SFRs of:

- Packet filtering rules: FDP_IFC.1, FDP_IFF.1
- Interpretation of network protocol: FIA_UID.1
- Maintenance of rules: FMT_MTD.1(NI)

7.3.4 Identification and Authentication

User identification and authentication in the TOE includes all forms of interactive login (e.g. using the SSH protocol or log in at the local console) as well as identity changes through the su and sudo commands. These all rely on explicit authentication information provided interactively by a user. In addition, the key-based authentication mechanism of the OpenSSH server is another form of of authentication.

7.3.4.1 PAM-based identification and authentication mechanisms

When a user possesses an identity in a system in the form of a login ID, that user has Identification. Identification establishes user accountability and access restrictions for actions on a system. Authentication is verification that the user's claimed identity is valid, and is implemented through a user password at login time.

All discretionary access-control decisions made by the kernel are based on the process's user ID established at login time and all mandatory access control decisions made by the kernel are based on the process domain established through login, which make the authentication process a critical component of a system.

The Linux system implements identification and authentication through a set of trusted programs and protected databases. These trusted programs use an authentication infrastructure called the Pluggable Authentication Module (PAM). PAM allows different trusted programs to follow a consistent authentication policy. PAM provides a way to develop programs that are independent of the authentication scheme. These programs need authentication modules to be attached to them at run-time in order to work. Which authentication module is to be attached is dependent upon the local system setup and is at the discretion of the local system administrator.

Linux uses a suite of libraries called the "Pluggable Authentication Modules" (PAM) that allow an administrative user to choose how PAM-aware applications authenticate users. The TOE provides PAM modules that implement all the security functionality to:

- Provides login control and establishing all UIDs, GIDs and login ID for a subject
- Ensure the quality of passwords
- Enforce limits for accounts (such as the number of maximum concurrent sessions allowed for a user)
- Enforce the change of passwords after a configured time including the password quality enforcement
- Enforcement of locking of accounts after failed login attempts.
- Restriction of the use of the root account to certain terminals
- Restriction of the use of the su and sudo commands

The login processing sets the real, file system effective and login UID as well as the real, effective, file system GID and the set of supplemental GIDs of the subject that is created. It is of course up to the client application usually provided by a remote system to protect the user's entry of a password correctly (e. g. provide only obscured feedback).

During login processing, the user is shown a banner. After successful authentication, the login time is recorded.

When configuring the OpenSSH server, the administrator is allowed to enable SSH key-based authentication in addition or instead of the username/password based authentication. When a user can successfully authenticate using the SSH key-based authentication based on a private SSH key in his possession, the TOE grants the user access.

SSSD is a system daemon with the primary function of providing access to identity and authentication remote resource through a common framework that can provide caching and offline support to the system. It provides PAM and NSS modules. It provides also a better database to store local users as well as extended user data. SSSD can be configured to use a native LDAP domain (that is, an LDAP identity provider with LDAP authentication), or an LDAP identity provider with Kerberos authentication. One of the primary benefits of SSSD is offline authentication. This solves the case of users having a separate corporate account and a local machine account because of the common requirement to implement a Virtual Private Network (VPN). SSSD can cache remote identities and authentication credentials. This means that a user can still authenticate with these remote identities even when a machine is offline. In an SSSD system, a user only needs to manage one account. SSSD integrates with the PAM and NSS framework and can therefore be used together with PAM modules for local credential stores.

After a successful identification and authentication, the TOE initiates a session for the user and spawns the initial login shell as the first process the user can interact with. The TOE provides a mechanism to lock a session either automatically after a configurable period of inactivity for that session or upon the user's request.

The TOE ensures that the memory used for the authentication operation is cleared before the authentication takes place. This ensures that previously entered credentials are not re-used for a new authentication operation.

When a new user is created, a complete new entry is added to the credential database. This ensures that previously existing credentials are not reused for the newly added user.

After successful authentication, a new process is spawned where the spawned process is identified by the "shell" entry in the credential store (either SSSD or /etc/passwd). This new process is spawned with the UID associated to the user in the credential store, In addition, the new process is spawned with the primary GID as well as supplemental GIDs defined by the credential store for the user (either SSSD or /etc/group). The capabilities are initially set as follows: if the UID of the user is 0, all capabilities are assigned to the newly spawned process. Otherwise no capabilities are assigned.

This security function covers the SFRs of FDP_RIP.2, FIA_AFL.1, FMT_MOF.1, FIA_UAU.1(HU), FIA_UAU.1(RITE), FIA_UID.1, FIA_UAU.5, FIA_UAU.7, FIA_USB.1.

Pluggable Authentication Module

PAM is responsible for the identification and authentication subsystem. PAM provides a centralized mechanism for authenticating all services. PAM allows for limits on access to applications and alternate, configurable authentication methods. For more detailed information about PAM, see the PAM project Web site at http://www.kernel.org/pub/linux/libs/pam.

PAM consists of a set of shared library modules, which provide appropriate authentication and audit services to an application. Applications are updated to offload their authentication and audit code to PAM, which allows the system to enforce a consistent identification and authentication policy, as well as to generate appropriate audit records. The following programs are enhanced to use PAM:

- login
- passwd
- su, sudo
- useradd, usermod, userdel
- groupadd, groupmod, groupdel
- sshd
- chage

- chfn
- chsh
- newrole

A PAM-aware application generally goes through the following steps:

1. The application makes a call to PAM to initialize certain data structures. With the initialization, the calling application provides a name to PAM which ultimately is used to find the configuration file of the authentication stack configuration in /etc/pam.d/. Usually, that name equals the application name.
2. The PAM module locates the configuration file for that application from /etc/pam.d/application_name and obtains a list of PAM modules necessary for servicing that application. If it cannot find an application-specific configuration file, then it uses /etc/pam.d/other.
3. Depending on the order specified in the configuration file, PAM loads the appropriate modules for the PAM operation requested by the calling application (i.e. PAM provides one call back for each module type – the module type is consistent with the “auth”, “session”, “password” and “account” sections in the PAM configuration files.
4. The authentication module code performs the requested operation depending on the module type. The module may require input from the user. Note: a module may perform operations which hardly have anything to do with authentication, but whose operations are necessary to set up the user environment.
5. Each authentication module performs its action and relays the result back to the application.
6. The PAM library is modified to create a USER_AUTH type of audit record to note the success or failure from the authentication module.
7. The application takes appropriate action based on the aggregate results from all authentication modules.

PAM modules

Linux is configured to use the following PAM modules – each PAM module used in the evaluated configuration is accompanied by a man page that provides additional information:

- pam_unix.so Supports all four module types, and provides standard password-based authentication. pam_unix.so uses standard calls from the system libraries to retrieve and set account information as well as to perform authentication. Authentication information about Linux is obtained from the /etc/passwd and /etc/shadow files. To perform the authentication, the pam_unix module calls the unix_chkpwd helper program.
- pam_stack.so pam_stack.so module performs normal password authentication through recursive stacking of modules. For example, the argument service=system-auth passed to the pam_stack.so module indicates that the user must pass through the PAM configuration for system authentication, found in /etc/pam.d/system-auth.
- pam_passwdqc.so Performs additional password strength checks. For example, it rejects passwords such as “1qaz2wsx” that follow a pattern on the keyboard. In addition to checking regular passwords it offers support for passphrases and can provide randomly generated passwords.
- pam_env.so Loads a configurable list of environment variables, and is configured with the /etc/security/pam_env.conf file.

- pam_shells.so Authentication is granted if the user’s shell is listed in /etc/shells. If no shell is in /etc/passwd (empty), then /bin/sh is used. It also checks to make sure that /etc/shells is a plain file and not world-writable.
- pam_limits.so This module imposes user limits on login. It is configured using the /etc/security/limits.conf file. No limits are imposed on UID 0 accounts.
- pam_rootok.so This module is an authentication module that performs one task: if the id of the user is 0, then it returns PAM_SUCCESS. With the “sufficient” control flag, it can be used to allow password-free access to some service for root.
- pam_xauth.so This module forwards xauth cookies from user to user. Primitive access control is provided by ~/.xauth/export in the invoking user's home directory, and ~/.xauth/import in the target user's home directory.
- pam_wheel.so Returns successful if the user to be authenticated is part of the wheel group. First, the module checks for the existence of a wheel group. Otherwise, the module defines the group with group ID 0 to be the wheel group.
- pam_nologin.so Provides standard UNIX nologin authentication. If the /etc/nologin file exists, only root is allowed to log in; other users are turned away with an error message (and the module returns PAM_AUTH_ERR or PAM_USER_UNKNOWN). All users (root or otherwise) are shown the contents of /etc/nologin.
- pam_loginuid.so Sets the audit uid for the process that was authenticated.
- pam_securetty.so Provides standard UNIX securetty checking, which causes authentication for root to fail unless the calling program has set PAM_TTY to a string listed in the /etc/securetty file. For all other users, pam_securetty.so succeeds.
- pam_faillock.so Keeps track of the number of login attempts made and denies access based on the number of failed attempts, which is specified as an argument to pam_faillock.so module. This is addressed at the “account” module type. The pam_faillock program allows administrative users to examine and control the pam_faillock PAM module's tally file, such as reset.
- pam_tally2.so Keeps track of the number of login attempts made and denies access based on the number of failed attempts, which is specified as an argument to pam_tally2.so module. This is addressed at the “account” module type. The pam_tally2 program allows administrative users to examine and control the pam_tally2 PAM module's tally file such as reset.
- pam_listfile.so Allows the use of ACLs based on users, ttys, remote hosts, groups, and shells.
- pam_deny.so Always returns a failure.
- pam_selinux.so The pam_selinux PAM offers role and level selection and sets the default security context for the session managed by PAM. pam_selinux prompts users to enter a non-default role and level, and calls into the SELinux libraries to obtain the default security context for the next executed shell in the session. It is not used for multilevel SSH sessions.
- pam_namespace.so Allows configuration of polyinstantiated directories using a per-session private namespace. A polyinstantiated directory provides an instance of itself to a process based on the user ID and/or the security context of the process. Directories to be polyinstantiated, location, and names of instance directories and the method used for polyinstantiation can be configured by modifying the /etc/security/namespace.conf file. The pam_namespace module relies on the per-session private namespace feature. pam_namespace invokes the unshare system call disassociates from the parent namespace, creates an instance directory as configured by the namespace.conf, assigns the security

context to what is returned by the security_compute_member library routine, and binds it on top of the directory to polyinstantiate. security_compute_member is an interface to the system security policy to query its type member rules.

- pam_cracklib.so The action of this module is to prompt the user for a password and check its strength against a system dictionary and a set of rules for identifying poor choices. The first action is to prompt for a single password, check its strength and then, if it is considered strong, prompt for the password a second time (to verify that it was typed correctly on the first occasion). All being well, the password is passed on to subsequent modules to be installed as the new authentication token.
- pam_sss.so The SSSD service is integrated into the PAM framework with the pam_sss.so module. Further information about SSSD is given below.
- pam_systemd.so pam_systemd registers user sessions with the systemd login manager systemd-logind.service(8), and hence the systemd control group hierarchy.

7.3.4.2 User Identity Changing

Users can change their identity (i.e., switch to another identity) using one of the following commands provided with the TOE:

su command

The su command is intended for a switch to a another identity that establishes a new login session and spawns a new shell with the new identity. When invoking su, the user must provide the credentials associated with the target identity - i.e. when the user wants to switch to another user ID, it has to provide the password protecting the account of the target user.

The primary use of the su command within the TOE is to allow appropriately authorized individuals the ability to assume the root identity to perform administrative actions. In this system the capability to login as the root identity has been restricted to defined terminals only. In addition the use of the su command to switch to root has been restricted to users belonging to a special group. Users that don’t have access to a terminal where root login is allowed and are not member of that special group will not be able to switch their real, file system and effective user ID to root even if they would know the authentication information for root. Note that when a user executes a program that has the setuid bit set, only the effective user ID and file system ID are changed to that of the owner of the file containing the program while the real user ID remains that of the caller. The login ID is neither changed by the su command nor by executing a program that has the setuid or setgid bit set as it is used for auditing purposes.

sudo command

The sudo command is intended for giving users permissions to execute commands with another user identity. When invoking sudo, the user has to authenticate with this credentials.

Sudo is associated with sophisticated ruleset that can be engaged to specify which:

- source user ID
- originating from which host
- can access a command, a command with specific configuration flags, or all commands within a directory
- with which new user identity.

When switching identities, the real, file system and effective user ID and real, file system and effective group ID are changed to the one of the user specified in the command (after successful authentication as this user).

Note: The login ID is not retained for the following special case:

1. User A logs into the system.
2. User A uses su to change to user B.
3. User B now edits the cron or at job queue to add new jobs. This operation is appropriately audited with the proper login ID.
4. Now when the new jobs are executed as user B, the system does not provide the audit information that the jobs are created by user A.

The su command invokes the common authentication mechanism to validate the supplied authentication.

This security function covers the SFRs of FIA_USB.1.

7.3.4.3 Authentication Data Management

Each TOE instance maintains its own set of users with their passwords and attributes. Although the same human user may have accounts on different servers interconnected by a network and running an instantiation of the TOE, those accounts and their parameter are not synchronized on different TOE instances. As a result the same user may have different user names, different user Ids, different passwords and different attributes on different machines within the networked environment. Existing mechanism for synchronizing this within the whole networked system are not subject to this evaluation.

Each TOE instance within the network maintains its own administrative database by making all administrative changes on the local TOE instance. System administration has to ensure that all machines within the network are configured in accordance with the requirements defined in this Security Target.

The file /etc/passwd contains for each user the user’s name, the id of the user, an indicator whether the password of the user is valid, the principal group id of the user and other (not security relevant) information. The file /etc/shadow contains for each user a hash of the user's password, the userid, the time the password was last changed, the expiration time as well as the validity period of the password and some other information that are not subject to the security functions as defined in this Security Target. Users are allowed to change their passwords by using the passwd command. This application is able to read and modify the contents of /etc/shadow for the user’s password entry, which would ordinarily be inaccessible to a non-privileged user process. Users are also warned to change their passwords at login time if the password will expire soon, and are prevented from logging in if the password has expired.

The time of the last successful logins is recorded in the directory /var/log/faillock where one file per user is kept.

The TOE displays informative banners before or during the login to users. The banners can be specified with the files /etc/issue for log ins via the physical console or /etc/issue.net for remote log ins, such as via SSH. When performing a log in on the physical console, the banner is displayed above the username and password prompt. For log ins via SSH, the banner is displayed to the remote peer during the SSH-session handshake takes place. The remote SSH client will display the banner to the user. When using the provided OpenSSH client, the banner is displayed when the user instructs the OpenSSH client to log into the remote system.

Users can change their own password. Only administrators can add or delete users or change their properties.

This security function covers the SFRs of FIA_ATD.1, FMT_REV.1(USR).

7.3.4.4 SSH key-based authentication

In addition to the PAM-based authentication outlined above, the OpenSSH server is able to perform a key-based authentication. When a user wants to log on, instead of providing a password, the user applies his SSH key. After a successful verification, the OpenSSH server considers the user as authenticated and performs the PAM-based operations as outlined above.

To establish a key-based authentication, a user first has to generate an RSA, DSA, or ECDSA key pair. The private part of the key pair remains on the client side. The public part is copied to the server into the file .ssh/authorized_keys which resides in the home directory of the user he wants to log on as. When the login operation is performed the SSHv2 protocol tries to perform the "publickey" authentication using the private key on the client side and the public key found on the server side. The operations performed during the publickey authentication is defined in [RFC4252] chapter 7.

Users have to protect their private key part the same way as protecting a password. Appropriate permission settings on the file holding the private key is necessary. To strengthen the protection of the private key, the user can encrypt the key where a password serves as key for the encryption operation. See ssh-keygen(1) for more information.

This security function covers the SFRs of FIA_UAU.1(HU), FIA_UAU.1(RITE), FIA_UID.1, FIA_UAU.5, FMT_MTD.1(CM).

7.3.4.5 Session locking

The TOE uses the screen(1) application which locks the current session of the user either after an administrator-specified time of inactivity or upon the user's request.

To unlock the session, the user must supply his password. Screen uses PAM to validate the password and allows the user to access his session after a successful validation.

This security function covers the SFRs of FTA_SSL.1, FTA_SSL.2.

7.3.5 Discretionary Access Control

The general policy enforced is that subjects (i.e., processes) are allowed only the accesses specified by the policies applicable to the object the subject requests access to. Further, the ability to propagate access permissions is limited to those subjects who have that permission, as determined by the policies applicable to the object the subject requests access to.

A subject may possess one or more of the following capabilities which provide the following exemptions from the DAC mechanism:

- CAP_DAC_OVERRIDE: A process with this capability is exempt from all restrictions of the discretionary access control and can perform any action desired. For the execution of a file, the permission bit vector of that file must contain at least one execute bit.
- CAP_DAC_READ_SEARCH: A process with this capability overrides all DAC restrictions regarding read and search on files and directories.
- CAP_CHOWN: A process with this capability is allowed to make arbitrary changes to a file's UID or GID.

- CAP_FOWNER: Setting permissions and ownership on objects even if the process' UID does not match the UID of the object.
- CAP_FSETID: Don't clear SUID and SGID permission bits when a file is modified.

DAC provides the mechanism that allows users to specify and control access to objects that they own. DAC attributes are assigned to objects at creation time and remain in effect until the object is destroyed or the object attributes are changed. DAC attributes exist for, and are particular to, each type of named object known to the TOE. DAC is implemented with permission bits and, when specified, ACLs.

The outlined DAC mechanism applies only to named objects which can be used to store or transmit user data. Other named objects are also covered by the DAC mechanism but may be supplemented by further restrictions. These additional restrictions are out of scope for this evaluation. Examples of objects which are accessible to users that cannot be used to store or transmit user data are: virtual file systems externalizing kernel data structures (such as most of procfs, sysfs, binfmt_misc) and process signals.

During creation of objects, the TSF ensures that all residual contents is removed from that object before making it accessible to the subject requesting the creation.

When data is imported into the TOE (such as when mounting disks created by other trusted systems), the TOE enforces the permission bits and ACLs applied to the file system objects.

During the creation of file system objects, the TOE ensures that new and zeroized memory is used for the newly allocated object. This ensures that any data previously present in the storage area is overwritten.

7.3.5.1 Permission bits

The TOE supports standard UNIX permission bits to provide one form of DAC for file system objects in all supported file systems. There are three sets of three bits that define access for three categories of users: the owning user, users in the owning group, and other users. The three bits in each set indicate the access permissions granted to each user category: one bit for read (r), one for write (w) and one for execute (x). Note that write access to file systems mounted as read only (e. g. CD-ROM) is always rejected (the exceptions are character and block device files which can still be written to as write operations do not modify the information on the storage media). The SAVETXT attribute is used for world-writable temp directories preventing the removal of files by users other than the owner.

Each process has an inheritable “umask” attribute which is used to determine the default access permissions for new objects. It is a bit mask of the user/group/other read/write/execute bits, and specifies the access bits to be removed from new objects. For example, setting the umask to “002” ensures that new objects will be writable by the owner and group, but not by others. The umask is defined by the administrator in the /etc/login.defs file or 022 by default if not specified.

This security function covers the SFRs of FDP_ACC.1(PSO), FDP_ACF.1(PSO), FDP_RIP.2, FMT_REV.1(OBJ), FMT_MSA.4.

7.3.5.2 Access Control Lists (ACLs)

The TOE provides support for POSIX type ACLs to define a fine grained access control on a user basis. ACLs are supported for all file system objects stored with the following file systems:

- ext4
- XFS

- tmpfs

An ACL entry contains the following information:

- A tag type that specifies the type of the ACL entry
- A qualifier that specifies an instance of an ACL entry type
- A permission set that specifies the discretionary access rights for processes identified by the tag type and qualifier

An ACL contains exactly one entry of three different tag types (called the "required ACL entries” forming the "minimum ACL"). The standard UNIX file permission bits as described in the previous section are represented by the entries in the minimum ACL.

A default ACL is an additional ACL which may be associated with a directory. This default ACL has no effect on the access to this directory. Instead the default ACL is used to initialize the ACL for any file that is created in this directory. If the new file created is a directory it inherits the default ACL from its parent directory. When an object is created within a directory and the ACL is not defined with the function creating the object, the new object inherits the default ACL of its parent directory as its initial ACL.

7.3.5.3 File system objects

Access to file system objects is generally governed by permission bits. For the above mentioned file system, ACLs are supported.

File system objects access checks are performed when the object is initially opened, and are not checked on each subsequent access. Changes to access controls (i.e., revocation) are effective with the next attempt to open the object.

7.3.5.4 Special Permissions

In addition, the following additional access control bits are processed by the kernel:

- SUID bit: When an executable marked with the SUID bit is executed, the effective UID of the process is changed to the UID of the owner of the file. The SUID bit for file system objects other than files is ignored.
- SGID bit: When an executable marked with the SGID bit is executed, the effective GID of the process is changed to the owning GID of the file. The SGID bit for file system objects other than files is ignored.
- SAVETXT: When a directory is marked with the SAVETXT bit, only the owner of a file system object in that directory can remove it. This bit is commonly used for world-writable directories like /tmp. Only processes with the CAP_FOWNER capability are able to remove the file system object if their UID is different than the owning UID of the file system object.

7.3.5.5 IPC objects

The System V IPC consists of message queues, semaphores, and shared memory regions. Message queues allow formatted data streams to be sent between processes. Semaphores allow processes to synchronize their execution. Shared memory segments allow multiple processes to share portions of their virtual address spaces.

This section describes data structures and algorithms used by the Linux kernel to implement the System V IPC. This section also focuses on the implementation of the enforcement of DAC, LSM decisions and handling of object reuse by the allocation algorithms.

The IPC mechanisms share the following common properties:

- Each mechanism is represented by a table in kernel memory whose entries define an instance of the mechanism.
- Each table entry contains a numeric key, which is used to reference a specific instance of the mechanism.
- Each table entry has an ownership designation and access permissions structure associated with it. The creator of an IPC object becomes its owner. This ownership can be transferred by the control system call of the IPC mechanism. The owner and root user are allowed to define and modify access permissions to the IPC object. Credentials of the process attempting access, ownership designation, and access permissions are used for enforcing DAC. The root user is allowed to override DAC setup through access permissions. In addition to the DAC permission information, a pointer security is maintained. This pointer is used by the active LSM. For SELinux, it contains the SELinux security context used for MAC enforcement.
- Each table entry includes status information such as time of last access or update.
- Each mechanism has a control system call to query and set status information, and to remove an instance of a mechanism.

Common data structures

The following list describes security-relevant common data structures that are used by all three IPC mechanisms:

- ipc_ids: The ipc_ids data structure fields, such as size, which indicates the maximum number of allocatable IPC resources; in_use, which holds the number of allocated IPC resources; and, entries, which points to the array of IPC resource descriptors.
- ipc_id: The ipc_id data structure describes the security credentials of an IPC resource with the p field, which is a pointer to the credential structure of the resource.
- kern_ipc_perm: The kern_ipc_perm data structure is a credential structure for an IPC resource with fields such as key, uid, gid, cuid, cgid, mode, seq, and security. uid and cuid represent the owner and creator user ID. gid and cgid represent the owner and creator group ID. The mode field represents the permission bit mask and the seq field identifies the slot usage sequence number. The security field is a pointer to a structure that contains the LSM security data structure which is a void pointer to allow LSMs to store data it needs.

IPC Access Control

The ipcperms function is called when a process attempts to access an IPC resource. ipcperms first enforces the DAC policy, and if DAC grants access, it calls security_ipc_permission to invoke the LSM for enforcing the LSM-specific policy. Discretionary access to the IPC resource is granted based on the same logic as that of regular files, using the owner, group, and access mode of the object. The only difference is that the owner and creator of the IPC resource are treated equivalently, and the execute permission flag is not used.

As the IPC objects of UNIX domain socket special files and Named Pipes are represented as file system objects, the access control mechanism covering file system objects are applicable to these IPC mechanisms too.

The TOE maintains IPC object types where each process has its own namespace for that object type: sockets - including network sockets. Access to the socket is only possible by the process whose socket namespace contains the socket reference. Setting of permissions for such objects can be handled using file descriptor passing.

The access control rules for IPC are identical to the ones available to files with the following differences:

- Special bits like SUID, SGID and SAVETXT do not exist.
- A process is given access to an IPC object irrespective of the permission settings if the process possesses the CAP_IPC_OWNER capability.

This security function covers the SFRs of FDP_ACC.1(TSO), FDP_ACF.1(TSO), FMT_REV.1(OBJ).

7.3.6 Security Management

The security management facilities provided by the TOE are usable by authorized users and/or authorized administrators to modify the configuration of TSF. The configuration of TSF are hosted in the following locations:

- Configuration files (or TSF databases)
- Data structures maintained by the kernel and within the kernel memory

The TOE provides applications to authorized users as well as authorized administrators to perform various administrative tasks. These applications are documented as part of the administrator and user guidance. These applications are either used to modify configuration files or to access parameters controlled and enforced by the kernel via kernel-provided interfaces to user space.

Configuration options are stored in different configuration files. These files are protected using the DAC mechanisms against unauthorized access where usually the root user only is allowed to write to the files. In some special cases (like for /etc/shadow), the file is even readable to the root user only. It is the task of the persons responsible for setting up and administrating the system to ensure that the access control features of the TOE are used throughout the lifetime of the system to protect those databases. These configuration files are accessed using applications which are able to interpret the contents of these configuration files. Each TOE instance maintains its own TSF database. Synchronizing those databases is not performed in the evaluated configuration. If such synchronization is required by an organization it is the responsibility of an administrative user of the TOE to achieve this either manually or with some automated assistance.

To access data structures maintained by the kernel, applications use the kernel-provided interfaces, such as system calls, virtual file systems, netlink sockets, and device files. These kernel interfaces are restricted to authorized administrators or authorized users, if applicable, by either using DAC (for virtual file system objects) or special kernel-internal verification checks for each interface.

7.3.6.1 Privileges

Privileges to perform administrative actions are maintained by the TOE. These privileges are separated into privileges to act on data or access functionality in user space and in kernel space.

Functionality accessible in user space are applications that can be invoked by users. Also, data accessible in user space is either data maintained with an application or data stored in persistent or transient storage objects. Privileges are controlled by permissions to invoke applications and to access data. For example, the configuration files including the user databases of /etc/passwd and /etc/shadow are accessible to the root user only. Therefore, the root user is given the privilege to perform modifications on this configuration data which constitutes administrative actions.

Functionality and data maintained by the kernel must be accessed using system calls. The kernel implements a privilege check for functions and data that shall not be accessible by normal users. These privileges are controlled with capabilities that can be assigned to processes. If a process is assigned with a capability, it is allowed to request special operations that other processes cannot. To implement consistency with the Unix legacy, processes with the effective UID of zero are implicitly given all capabilities. However, these processes may decide to drop capabilities. Such capabilities are marked by names with the prefix of "CAP_" throughout this document. The Linux kernel implements many more capabilities than mentioned in this document. These unmentioned capabilities protect functions that do not directly cover SFR functionality but need to be protected to ensure the integrity of the system and its resources.

The TOE provides security management applications for all security-relevant settings listed throughout this ST, i.e. all FMT_MSA.1, FMT_MSA.3 and FMT_MTD.1 iterations, FMT_SMR.1.

8 Abbreviations, Terminology and References

8.1 Abbreviations

ACL	Access Control List
API	Application Programming Interface
KVM	Kernel Virtualized Machine
HTTP	Hypertext Transfer Protocol
SFR	Security Functional Requirement
SSL	Secure Sockets Layer
ST	Security Target
TCP/IP	Transmission Control Protocol / Internet Protocol
TLS	Transport Layer Security
TOE	Target of Evaluation
TSF	TOE Security Functionality
VM	Virtual Machine
VPN	Virtual Private Network

8.2 Terminology

This section contains definitions of technical terms that are used with a meaning specific to this document. Terms defined in the [CC] are not reiterated here, unless stated otherwise.

Authentication Data

Authentication data is the data used by users or remote entities to authenticate their claimed identity.

Authorized Administrator
This term refers to a user in one of the defined administrative roles of a Linux system. The TOE associates the user with the UID of zero and named "root" with administrative authorities. Effectively, the UID zero is assigned with all Linux capabilities known to the Linux kernel. Every user who is allowed to log on as that root user, or to switch their UID to the root user is considered an authorized administrator. In addition, any user who is able to execute applications which grant one or more Linux capabilities to be used in an unconditional manner is considered an authorized administrator. Note: the process executing on behalf of the root user must possess MLS override attributes to perform management aspects of the Mandatory Access Control Policy.

Category
A category is the non-hierarchical category of the lower MLS label defined with an SELinux label. Note: an SELinux label consists of four parts where the MLS label is one of them. The MLS label in turn is split into a higher and a lower MLS label part.

Classification
A sensitivity label associated with an object.

Clearance
A sensitivity label associated with a subject or user.

DAC
Discretionary Access Control implemented with permission bits and ACLs.

Data
Arbitrary bit sequences on persistent or transient storage media.

Dominate
Sensitivity label A dominates sensitivity label B if the hierarchical level of A is greater than or equal to the hierarchical level of B, and the category set of label A is a proper subset of or equal to the category set of label B. (cf. Incomparable sensitivity labels).

Guest
Software executing within a virtual machine environment. There can be zero or more guests executing concurrently on the host system.

Host
The host system provides the Linux environment that controls and manages the virtual machines. The host provides the execution environment for every virtual machine.

Information
Any data held within a server, including data in transit between systems.

IOMMU
Input / Output Memory Management Unit. This MMU allows the setup of multiple DMA areas for different virtual machines.

KVM
Kernel-based Virtual Machine.

MLS
Multi-level security

Named Object
In Linux, those objects that are covered by access control policies. The list of objects defined as named objects is provided with FDP_ACC.1.

Object
For Linux, objects are defined by FDP_ACC.1.

OSPP
Operating System Protection Profile

OSPP EP
Operating System Protection Profile Extended Package

PAM
Pluggable Authentication Module - the authentication functionality provided with Linux is highly configurable by selecting and combining different modules implementing different aspects of the authentication process.

Product
The term product is used to define software components that comprise the Linux system.

QEMU
The QEMU software component implements the virtual devices and virtual resources for virtual machines. There is one instance of QEMU per virtual machine. The QEMU software component is also identified as the "kvm" application on the host system.

SELinux
Linux kernel LSM module that is able to implement arbitrary security policies. An SELinux policy distributed with the TOE implements multi-level or multi-category security.

Sensitivity Label
The TOE attaches a sensitivity label to each named object. This label consists of a hierarchical sensitivity level and a set of zero or more categories. The policy defines the number and names of the sensitivity levels and categories.

Subject
There are two classes of subjects in Red Hat Enterprise Linux: i) untrusted internal subject - this is a Linux process running on behalf of some user or providing an arbitrary service, running outside of the TSF (for example, with no privileges); ii) trusted internal subject - this is a Linux process running as part of the TSF (for example: service daemons and the process implementing the identification and authentication of users).

Target Of Evaluation (TOE)
The TOE is defined as the Red Hat Enterprise Linux operating system, running and tested on the hardware and firmware specified in this Security Target. The BootPROM firmware as well as the hardware are not part of the TOE.

User
Any individual/person or technical entity (such as a service added by the administrator on top of the TOE) who has a unique user identifier and who interacts with the product.

User Security Attributes
Defined by functional requirement FIA_ATD.1, every user is associated with a number of security attributes which allow the TOE to enforce its security functions on this user. This also includes the user clearance which defines the maximum sensitivity label a user can have access to.

Virtual devices
See virtual resources for a generic explanation. This definition applies also to virtual devices, but with a focus to devices, such as disks, network cards, graphics cards, and similar.

Virtual machine
A virtual machine is an execution environment where the software executing within the virtual machine has access to the processor's user and supervisor state and resources defined by the host system. Resources include the number of processors, RAM size, physical devices, virtualized devices, communication channels to other virtual machines and the host system. For the KVM environment a virtual machine environment is controlled and provided by the Linux kernel hypervisor functionality plus the QEMU application instantiated for each virtual machine.

Virtual machine environment
See virtual machine.

Virtual resources
Virtual resources are resources that either do not physically exist and do not exist in the host system. Virtual resources are implemented by the virtual machine environment and are provided to the respective virtual machine. For example, virtual resources are special exceptions that can be triggered from the virtual machine environment to request services from the host system, such as para-virtualized drivers. Virtual devices can be considered one form of virtual resources.

8.3 References

CC	**Common Criteria for Information Technology Security Evaluation**	
	Version	3.1R4
	Date	September 2012
	Location	http://www.commoncriteriaportal.org/files/ccfiles/CCPART1V3.1R4.pdf
	Location	http://www.commoncriteriaportal.org/files/ccfiles/CCPART2V3.1R4.pdf
	Location	http://www.commoncriteriaportal.org/files/ccfiles/CCPART3V3.1R4.pdf
GPOSPP1	**General-Purpose Operating System Protection Profile Part 1**	
	Version	3.9
	Date	2012-12-06
GPOSPP2	**General-Purpose Operating System Protection Profile Part 2**	
	Version	3.9
	Date	2012-12-06
IPTABLES	**IPTables man page chapter 8**	
	Version	RHEL 7.1
	Date received	2015-08-01
IPTABLES-EXT	**IPTables Extensions man page chapter 8**	
	Version	RHEL 7.1
	Date received	2015-08-01
OSPP	**General-Purpose Operating System Protection Profile**	
	Version	3.9
	Date	2012-12-06

RFC4252	**The Secure Shell (SSH) Authentication Protocol**	
	Date	January 2006
	Location	http://tools.ietf.org/html/rfc4252
RFC4253	**The Secure Shell (SSH) Transport Layer Protocol**	
	Date	January 2006
	Location	http://tools.ietf.org/html/rfc4253
RFC4352	**RTP Payload Format for the Extended Adaptive Multi-Rate Wideband (AMR-WB+) Audio Codec**	
	Author(s)	J. Sjoberg, M. Westerlund, A. Lakaniemi, S. Wenger
	Date	2006-01-01
	Location	http://www.ietf.org/rfc/rfc4352.txt